DOCUMENT RESUME

ED 152 328 IR 005 880

AUTHOR Brown, David, C.; Kwasny, Stan C.
TITLE A Natural Language Graphics System.
INSTITUTION Ohio State Univ., Columbus. Computer and Information Science Research Center.
REPORT NO OSU-CISRC-TR-77-8
PUB DATE Jun 77
NOTE 68p.

EDRS PRICE MF-$0.83 HC-$3.50 Plus Postage.
DESCRIPTORS Artificial Intelligence; *Computer Graphics; Input Output; *Programing Languages; Systems Analysis
IDENTIFIERS LISP Program; *Natural Language

ABSTRACT

This report describes an experimental system for drawing simple pictures on a computer graphics terminal using natural language input. The system is capable of drawing lines, points, and circles on command from the user, as well as answering questions about system capabilities and objects on the screen. Erasures are permitted and language input can be embellished with touches to convey positional information. Written in LISP 1.6 for the DEC System-10 computer, the system was implemented to test out ideas on system organization, to establish the viability of combining language and graphics, and to experiment with appropriate artificial intelligence techniques. (Author/DAG)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED152328

U.S. DEPARTMENT OF HEALTH, EDUCATION & WELFARE
NATIONAL INSTITUTE OF EDUCATION

THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL NATIONAL INSTITUTE OF EDUCATION POSITION OR POLICY

OSU-CISRC-TR-77-8

A NATURAL LANGUAGE GRAPHICS SYSTEM

BY

David C. Brown

Stan C. Kwasny

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY

Celianna Taylor

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC) AND USERS OF THE ERIC SYSTEM."

Computer and Information Science Research Center
The Ohio State University
Columbus, Ohio 43210

June, 1977

I 200 5-880

PREFACE

The Computer and Information Science Research Center of The Ohio State University is an interdisciplinary research organization consisting of staff, graduate students, and faculty of many University departments and laboratories. This report describes research undertaken in cooperation with the Department of Computer and Information Science.

During the summer of 1976 when this research was performed, the two authors were supported by the T.E. French Fellowship and the Department of Computer and Information Science. These two sources also provided computing funds and technical support.

Partial support by Grant AFOSR 72-2351, B. Chandrasekaran, Principal Investigator, is gratefully acknowledged.

ACKNOWLEDGEMENTS

The authors would like to acknowledge the help and support of the following:

a. Dr. B. Chandrasekaran and Dr. N. K. Sondheimer for their constant encouragement, valuable ideas, and constructive criticism during this work;

b. Dr. H. W. Buttelmann and Dr. A. P. Lucido for their contributions during the initial discussions of the project;

c. Dr. S. C. Shapiro for allowing us to use a version of the MENTAL package;

d. Nick Eastridge for helping us with a number of details for the LISP-FORTRAN interface for the DEC System 10, and for providing us with some software; and

e. Dave Rypka for assistance with the programming of the LISP-FORTRAN interface, and as a source of Plasma Panel folk-lore.

0. CONTENTS:

PAGE

1. INTRODUCTION 1
   1.1 Overview
   1.2 Example of Dialogue
   1.3 Motivation
   1.4 Report Outline

2. OVERALL SYSTEM DESIGN 3
   2.1 General Description
   2.2 System Construction and Start-up
   2.3 The Executive Module
   2.4 The Error Module
   2.5 The Initialization Module
   2.6 Typical Control Flow

3. THE LANGUAGE ANALYSIS MODULE 8
   3.1 Function and Overall Design
   3.2 The Two Input Modules
   3.3 Parsing
      3.3.1 The Augmented Transition Network Model
      3.3.2 Lexical Entries
      3.3.3 The Grammar
   3.4 Representation of the Parse - the MG Interface
      3.4.1 Valid Case Forms
      3.4.2 Creating the Prototype
   3.5 The QA Interface
   3.6 Problems in LA

4. THE LANGUAGE GENERATION MODULE 16
   4.1 Function and Overall Design

5. THE KNOWLEDGE BASE MODULE 18
   5.1 Function and Overall Design
   5.2 The Semantic Memory
      5.2.1 Structure
      5.2.2 Actions
      5.2.3 Naming and Erasing

5.3 Question Answering
5.3.1 Introduction
5.3.2 The INFO Function
5.3.3 The FINDOBJ Function
5.3.4 The CONVERT Function
5.3.5 The FINDDRAW Function
5.4 Memory and Graphics
5.4.1 Introduction
5.4.2 Drawing
5.4.3 Erasing
5.4.4 Naming
5.5 Problems and Extensions
5.5.1 Moving
5.5.2 Remembering
5.5.3 Defaults
5.5.4 Structural Information
5.5.5 Defined Points
5.5.6 Point Notation
5.5.7 Screen Model and Action History
5.5.8 Functions

6. THE GRAPHICS MODULE 31
6.1 Introduction
6.2 The Screen Functions
6.3 Problems and Extensions

7. CONCLUSION 34
7.1 System Performance
7.1.1 General
7.1.2 Good Things
7.1.3 Bad Things
7.2 Outstanding Problems
7.3 The Future
7.4 Summary

8. REFERENCES 41

9. APPENDICES 43

1. INTRODUCTION

1.1 Overview

This report describes an experimental system for drawing simple pictures on a computer graphics terminal using natural language input. The system is capable of drawing lines, points, and circles on command from the user, as well as answering questions about system capabilities and objects on the screen. Erasures are also permitted. Language input can be embellished with touches to convey positional information.

The system was designed and implemented by the authors during Summer 1976, was written in LISP 1.6, runs in about 40K words on a DECSystem-10 computer, and displays pictures on an ag6Ø Plasma Panel.

The system was implemented tc test out ideas on system organization, to establish the viability of combining language and graphics, and to experiment with appropriate A.I. techniques.

1.2 Example of dialogue

```
*? PLEASE DRAW A VERTICAL 2 INCH LINE HERE <T>.
OK
*? PUT A POINT CALLED FRED HERE <T>.
OK
*? MAKE A CIRCLE WITH A TEN CM DIAMETER AT FRED.
OK
*? CONNECT FRED AND (100,150)
OK
*? CALL THE CIRCLE BALL
OK
*? ERASE THE LINE FROM FRED
OK
*? ERASE FRED
OK
*? CALL THE 2 INCH LINE BAT.
OK
*? DRAW A CIRCLE
OK
*? NAME THE CIRCLE FACE
OK
*? WHAT DID YOU DRAW HERE <T>?
a LINE called BAT
*? WHAT CAN YOU DRAW?
LINES, POINTS, and CIRCLES
*? IS THERE A CIRCLE CALLED FACE?
yes
*? CAN YOU DRAW SQUARES?
no
```

The system prompt is "*?", and the system response is on the following line. The symbols "<T>" indicate a touch on the screen. This extract

shows some of the variety of sentential forms available in the system. Not shown, are the results of the commands on the screen.

1.3 Motivation

The research being done by our group promises to be one more step towards the goal of natural interaction between man and computer. Our work is based on the belief that use of more than one mode of communication is required to achieve that goal. We have chosen to investigate the combination of natural language and graphics. A system using this combination allows the use of linguistic, graphical, or mixed forms for both input and output of information. Careful development of this idea would provide practical systems with a high degree of habitability.

Many artists and designers who are (and wish to remain) naive about programming will be able to interact productively with a natural language graphics system and such a system would allow a much wider group of people to use computers. Special subsets of both language and pictures can be developed for various uses. We are advocating a form of natural language programming, but with an additional (graphical) mode of communication. We feel that the feedback provided by the graphics will assist the user in detecting, and then interactively correcting, errors due to vagueness or ambiguity. In addition, the user is able to select the appropriate method for input of information and, for example, could provide a rough drawing of an object with a description of additional details. These techniques would be viable in a variety of applications areas, such as Animation, Architecture, Engineering, and Education.

Natural Language Graphics (NLG) provides a framework for research in several areas of computer science. It supplies a domain for the study of linguistic phenomena and language understanding system design. Memory and inference will play a large role in an NLG system and consequently knowledge representation and manipulation is important. Such systems would also lead to new ideas about Computer Grpahics, helping to liberate it from its present algorithmic approach. Man-machine interaction studies and AI systems organization will also need to be pursued.

There has been no systematic study of the combination of language and graphics, although there have been a few systems with this combination [Kirsch, Coles, Simmons, Badler and Winograd] and some recent work on graphical information [Sondheimer, Palmer, Agin, Nevatia and Minsky].

1.4 Report Outline

The next section, 2, is concerned with the overall design and operation of the system. Sections 3 and 5 describe the language processing and knowledge components respectively. As these are the longest and most important components they are both described in some detail. The language generation component is presented in Section 4, while graphical output is described in Section 6. The final section is a summary and conclusion.

2. OVERALL SYSTEM DESIGN

2.1 General Description

NLG is composed of nine independent modules. Communication among modules is achieved by a message-passing scheme in which any module may invoke another by sending an appropriate message. As illustrated in Figure SYS-1, all messages are handed to the Executive module (EXEC) which ultimately relays the message to its destination. Thus the system is organized heterarchically, although a specific calling sequence has been induced on the model which limits the interaction of modules in actual operation.

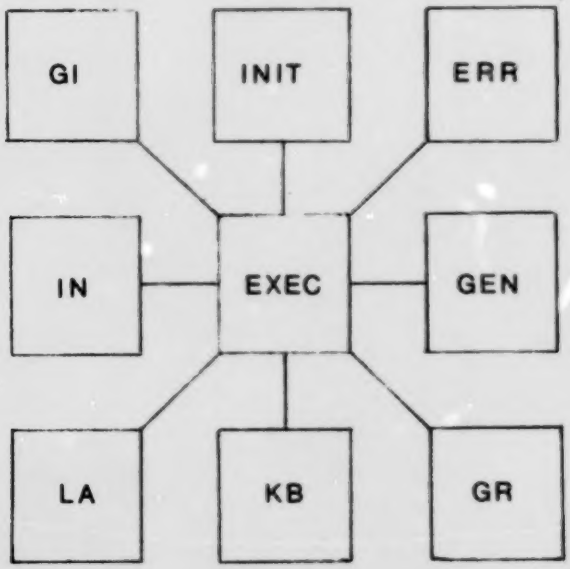


FIGURE SYS-1
System Organization

Each of the remaining eight modules has specific duties in the operation of the system. The Initialization module (INIT) contains startup information for NLG. If the system detects internal inconsistencies, the System Error module (ERR) is called to interact with the programmer in debugging. Modules for graphical (touch) input (GI) and typed text input (IN) provide preprocessed user input for language analysis. The Language Analysis module (LA) parses the input, using an ATN grammar, into a case-based semantic structure. The Knowledge Base module (KB) develops a semantic network from the input and creates instructions on how to update the display screen. The Graphics Output module (GR) uses these instructions to run a graphics program. Responses to the user are produced by the Language Generation module (GEN) using a generative ATN grammar.

NLG is written in LISP 1.6, except for the graphics primitive routines which are coded in MACRO-10. As a debugging feature, the evaluation of a LISP expression is permitted whenever the system prompts the user for input by preceding the expression with a dollar sign character ($).

Each module or collection of data used in building the system is contained on a separate file. (A complete breakdown of the size of each module appears in Appendix A.) In addition, an assembly of utility functions and functions useful in more than one module is kept on a single file. The original motivation for doing this was to facilitate the dynamic swapping of modules to maximize the use of memory. Although the mechanisms for this were implemented, processing time increased prohibitively. However, the complete separation of modules in this fashion eliminated many of the problems typically encountered in developing large systems.

2.2 System Construction and Startup

The system exists in LISP and MACRO-10 form. For ease of construction, a special control file is submitted to the monitor to build the system and create a core-image "SAVE" file. The content of the control file follows:

```
.COMP LARITH,ILSPF4,PLASUB
.R LISP 38;/A
1200 12400
*(SETQ *TRACEOFF* T)
*(PUTSYM FLONUM FIXNUM)
*(LOAD NIL)
*LARITH,ILSPF4,PLASUB $
*(DSKIN (NLGLP2 . LSP))
*(DECIMAL)
*(GC)
.SAV PLASMA.SAV
.DEL PLASMA.HGH
.K/F
```

where $ represents an altmode character.

First, the three MACRO-10 files are compiled: the file LARITH establishes the interface between LISP and the FORTRAN library arithmetic routines (i.e., SIN, COS, etc.); the file ILSPF4 builds the interface from LISP to the graphics primitives package; and the file PLASUB contains the plasma panel subroutines. LISP then runs in 38K with the full word space set at 1200 words (octal) and the free storage space set at 12400 words (octal). These values were established mostly through experimentation, but see the LISP reference manual [Quam and Diffie] for further details. Once in LISP, the KB trace mechanism, used for debugging, must be switched off by setting the flag *TRACEOFF*. The two internal LISP routines FLONUM and FIXNUM, which deal with converting fixed and floating numbers, must be put on the DDT symbol table to be available for the interfaces. Next, the relocatable files compiled earlier are loaded into expanded core. LISP makes use of the system loader which itself is loaded on call and unloaded when finished. At this point, the total core allocation for NLG may reach 43K. The system is loaded next. The file NLGLP2.LSP contains the individual file names along with their location on disk. Lastly, the system is placed in decimal mode and garbage collected. The complete system is now saved in the core-image file "PLASMA" and the task is completed. The sketchy description given above may be clarified singificantly be referring to the LISP manual's sections on adjusting system parameters and loading binary files.

Once a core image of NLG is available, a user types

```
.RU PLASMA
```

to gain access to it. At the prompt, a valid message to INIT may be sent via EXEC; however, for convenience, one may type

```
(INIT)
```

to establish contact with NLG.

2.3 The Executive Module

The Executive module (EXEC) receives and sends all messages in the system. A validity test is performed on messages according to the following syntax[†].

[†]The notation used in expressing syntax in this report is a modified BNF where the symbol "or" shows alternatives and subscripts following non-terminals show the minimum number of occurrences.

```
<message>  ::=  ( <to> <from> <mark> <args> )
<to>       ::=  <module name>
<from>     ::=  <module name>
<mark>     ::=   #Q  or  #R
<args>     ::=  ( <arg>* )
```

The module names used in the <to> and <from> components must correspond to the names given in Figure SYS-1. The list of arguments is not checked by EXEC, since each module generally protects itself from bad data.

Messages with a query mark (#Q) are sent as module calls while those with a response mark (#R) are returned as a value to the caller. Also, EXEC may be given the message by calling or returning. This allows the conveyance of messages in four ways, corresponding to the manipulation of the calling level in LISP and the calling level of NLG. The four possibilities are:

(i) calling EXEC with a #Q message;
(ii) calling EXEC with a #R message;
(iii) returning a #Q message to EXEC; and
(iv) returning a #R message to EXEC.

The first combination results in a conventional module call while the last results in a conventional return. The second possibility is never used in NLG. The third one allows module invocation in a manner similar to co-routining. LA and KB occasionally communicate in this fashion.

EXEC's final duty is to automatically produce a trace of all system messages on the file MESS.LPT, including statistics on time and space utilization.

2.4 The Error Module

The System Error module (ERR) exists solely as a debugging feature. If the message validity test conducted by EXEC fails, then ERR is called. It displays the bad message and interacts with the user to create a new one. Of course, the dollar sign ($) feature which allows LISP evaluation is supported allowing a knowledgeable person to completely investigate the problem. ERR may be considered an extension to EXEC, but has been rarely used.

2.5 The Initialization Module

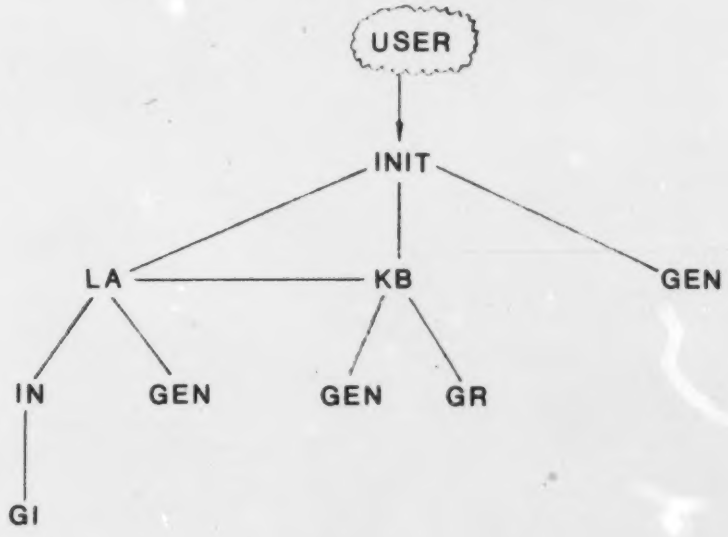
The system is activated with a user message to the Initialization module (INIT). Here, all global system parameters are initialized, output channels are established, and an initial screen erase is performed. The screen's cursor position is set to zero and the distance between lines

of text on the screen is set to 15 screen units. These parameters are utilized in repositioning the cursor after drawing. An introductory system response is then made via GEN.

Next, a message is sent to activate LA for input processing. It must be determined from the eventual reponse to this message whether to stop the system or to re-activate LA. For the former, an exit response is made and NLG halts, otherwise GEN is directed to respond "OK" and LA is called again.

2.6 Typical Control Flow

Although NLG is based on a heterarchical model, hierarchical relationships exist among modules viewed in operation. The following diagram illustrates typical control flow in the system†:



When INIT is activated by the user, it knows to expect input. Thus, LA is called. From there, IN is activated to process the raw input, with GI being called to handle graphical input. During LA's processing, KB may be consulted for stored or acquired knowledge. Upon completion, LA

†Of course, EXEC controls the passing of messages, but the diagram would become unnecessarily complicated if EXEC were included everywhere. The remainder of this paper will contain no mention of EXEC unless essential to the discussion.

can either pass control to KB or return to INIT. The former happens for commands and the latter for question-answering. When KB gets control, it concludes the processing by building memory structures and directing GR to draw. KB then returns to INIT. GEN may be activated by either INIT, LA, or KB.

3. THE LANGUAGE ANALYSIS MODULE

3.1 Function and Overall Design

The Language Analysis module (LA) processes all input to the system, including both typed text and touches to the screen. Figure LA-1 shows the logical structure of LA.

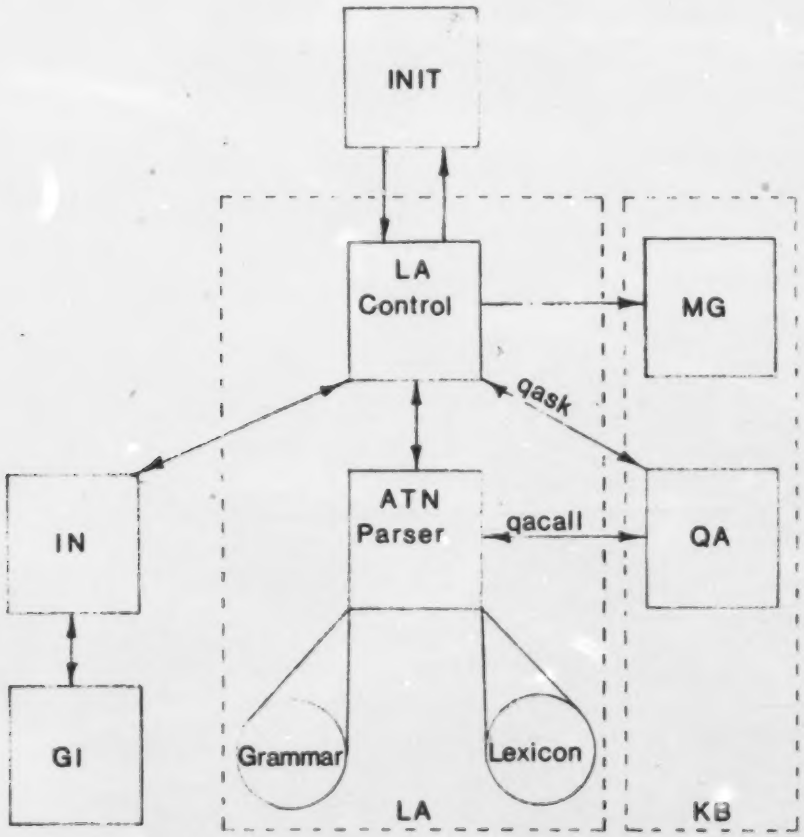


FIGURE LA-1
LA Organization

An Augmented Transition Network (ATN) parser [Woods, 1973] serves as the primary component of the module, and a call of LA may optionally specify an ATN state name or configuration. While parsing, the grammar permits queries to the Question-Answering (QA) mechanism within the

Knowledge Base (KB), via the function QACALL, to retrieve other stored knowledge as required. The parse results in a case-like structure called a prototype which contains a representation of the original input (see Section 3.4). If the input posed a question, the answer may then be retrieved from QA via the function QASK. For commands, the prototype is passed as the argument in a message to the Memory and Graphics (MG) mechanism within KB. If the input commanded the system to stop, then a return message is conveyed to INIT which stops the system.

3.2 The Two Input Modules

Before parsing can begin, the input must be scanned and assembled into list form. The Input module (IN) is designed for this purpose. If IN should detect a touch control character while scanning, then the Graphics Input module (GI) is activated to interpret a touch. Touches become screen coordinate pairs (a LISP dotted pair of numbers) in the input list. Encountering a carriage-return causes IN to return the input list to LA for parsing.

3.3 Parsing

3.3.1 The Augmented Transition Network Model

The ATN parser used in this module is modeled after the one used in the LUNAR system [Woods, 1973; Woods, et al., 1972]. A finite state network of states and arcs is endowed with recursive capabilities allowing a sub-network traversal to be performed in the course of following one arc. This recursive transition network is further augmented by the capacity to manipulate "registers" and perform function calls during traversals. Such registers can contain flags, words from the input string, partial parse trees, case structures, or any expression permitted by the host language. This allows the various constituents to be built, tested, altered, and steered into position within the representation chosen. The behavior of the parser is determined by the grammar and lexicon provided. The syntax of grammar specification in an ATN appears in Appendix B. The lexicon consists of a list of feature-value pairs for each word sense.

3.3.2 Lexical Entries

In NLG, new features were added to lexical entries throughout

the project as they become justifiable after discovering additional distinctions to be made while processing. Four features are considered basic to the parsing scheme: the syntactic category of the word sense (CTGY); the root form of the entry (ROOT); the number category (SING or PLUR) for nouns (NUM); and the transitivity of verbs (TRANS). A case feature (CASE) was also used to suggest which case or cases the entry might fit. For the position case, an indicator (TAG) showed the type of position being described by taking the values ENDPOINT or LOCATOR. For verbs like "connect" which literally means "draw line", the object to be drawn (i.e., a straight line) is specified under the feature OBJ. Words such as "circumference" or "diameter" which apply only to circles have the feature %OBJECT with a value of (CIRCLE) to indicate this dependence. Similarly, "degrees" applies as a size measurement to angles and is indicated by a SIZEOF feature with the proper value. Adjectival relationships are utilized in disambiguation to limit the number of possible word senses to consider. For example, a line can be STRAIGHT or CURVED and the feature ADJ is used to specify this fact. Determiners can be definite or indefinite indicated by a DEF feature with either a T or NIL value. Lastly, abbreviations are distinguished by the feature ABBREV. The input vocabulary appears in Appendix C while a partial lexicon is given in Appendix D.

3.3.3 The Grammar

The grammar for parsing can be viewed as five distinct networks: sentence, touch, noun phrase, prepositional phrase, and question. The five network diagrams appear in Appendices E, F, G, H, and I.

The sentence level network controls the construction of imperative prototypes. The initial state (S*) contains arcs which decide whether a command or a question form is present. Questions typically begin with auxiliaries or question words of various types and these are detected by the function QSTART, called as the test portion of the arc which looks for questions. Notice that the grammar permits prepositional phrases or touches to the screen at the beginning of a sentence. These are remembered using the HOLD mechanism until their proper place can be found later in the string. The imperative form is parsed from the state S*IMP. If the system is being commanded to stop, then the arc to state S*STOP builds the proper message to be conveyed to INIT and parsing stops. The arc to VP*HEAD is crucial in the development of the imperative

prototype. It finds the verb near the front of the sentence and establishes the prototype. At state VP*HEAD, the post-verbial constituents are expected. In most cases, a noun phrase will follow, but other possibilities remain. For verbs like "connect", a virtual object (ST-LINE) is implied. This will be followed by an indication of what two things are to be connected. Once at the state VP*, any of a number of constituents can be handled. In particular, a touch is treated directly in the grammar in a manner similar to noun phrases and prepositional phrases. When the sentence has been completely parsed, the resulting prototype structure is returned as the final value.

The touch network allows the parsing of a touch to the screen which has been preprocessed by GI, or equivalently, a specification in the input string of a pair of screen coordinates. Either of these may be optionally preceded by "here", "there", "this", or "that". After a touch point has been parsed, the LISP dotted pair of x-y coordinates is returned as the result. An arc has been provided at the accepting state of the touch network to recognize multiple touches to the same point on the screen. This became necessary after such a phenomenon was observed in operating the system. Of course, only one pair of coordinates is returned in such cases.

The noun phrase network describes the processing of entities used in the system as noun phrases. In addition to the standard noun phrase consisting of the head noun with nominal modifiers, this network also handles:

(1) names - e.g., "the point P" or simply "P";
(2) the preposition "of" - e.g., "an angle of 32 degrees" or "a line of length two inches";
(3) a touch used as a complementizer - e.g, "the line <T>"; and
(4) calls via QACALL to establish the identity of a name (FINDOBJ) or to change numeric scale (CONVERT).

Sufficient flexibility is provided in the network to process most reasonable combinations of the lexical items known to the system. The reader is referred to the NP* network in Appendix G for a more detailed exposition.

The prepositional phrase network provides an additional flexibility in processing. The preposition is found first. This can be either a standard preposition (e.g., to, from, at, through, etc.) or one that takes

a compound form (e.g., between). Next, either a noun phrase or a touch can be found. Finally, if a compound form is expected, as in "between the point P and here <T>", then this is parsed. Note that by using the HOLD mechanism and a VIR arc, the interpretation given is literally "between the point P and between here <T>". When complete, the phrase is returned and a partial prototype, built from the phrase, is lifted to the calling level.

The final component of the grammar is the question network. This has been developed to the point where several types of questions useful in NLG can be processed correctly, but remains somewhat incomplete as a general question-answering grammar. Nevertheless, the following questions result in answers as indicated:

(1) What can you draw?
LINES, POINTS, and CIRCLES

(2) Can you draw circles?
yes

(3) Can you draw a circle with a 3 inch circumference?
yes

(4) How many screen units is an inch?
64

(5) Is there a point named P?
yes

(6) Is there a circle named Fred here <T>?
no

(7) Is there a straight circle?
I don't know

The question network permits a variety of syntactic forms as shown in the diagram in Appendix I, but answers cannot be generated for some of the possible paths. This situation was allowed to develop intentionally, since the system is not primarily concerned with this capability. More work in this area is anticipated.

The five networks combine to form a grammar for NLG which performs well enough to parse most sentences in less than a half second - a major factor in allowing real time response. Appendix J contains some sample sentences with their timings.

3.4 Representation of the Parse - the MG Interface

The result of a completed parse is a structure called a prototype. Appendix K contains a description of one which shows the form permitted by each of the slots in the case-like structure. The function of each of these is discussed briefly below.

3.4.1 Valid Case Forms

The %TYPE case indicates what form of input was detected during parsing. Two types are distinguished by the system, namely the imperative (IMP) form and the question (Q) form. Only imperative forms result in calls on MG in the current implementation. Question forms are handled completely within LA.

Three actions are possible for the %ACTION case. These are drawing, erasing, and naming. The verb of the imperative sentence is reduced to one of these basic forms.

Only three objects are recognized by the system. Thus, the %OBJECT case can be filled by either ST-LINE, CIRCLE, or POINT.

The %NAME case is included to contain the name of the drawn or erased object. This name is limited in form to a LISP atom and is specified by the user in the input string.

The orientation of a line can be specified in the %ORIENT case. This contains a numeric value in degrees measured counter-clockwise from the horizontal.

Position on the screen can be indicated in the %POSIT case. Any number of points (usually no more than two) may be specified. Each point specification consists of a tag for either ENDPOINT, LOCATOR, or TOUCH, and the coordinates of the actual point. ENDPOINT is used to indicate the end points of a line. The midpoint of a line, the center of a circle, and the location of a point are all indicated by the tag LOCATOR. A TOUCH tag is used whenever the function of the point is not determined from the input sentence (e.g., "Erase this <T>"). The coordinates can be specified in actual screen units as a dotted pair of numbers, or they may be given by referring to the node identifier for a particular point stored in the KB. The latter is indicated by the tag NODE dotted to the node identifier.

Finally, the %SIZE case indicates the length of a straight line or the radius of a circle in screen units. In addition, by including one of the tags RADIUS, DIAMETER, or CIRCUMFERENCE, the size of a circle can be given in terms of one of these.

3.4.2 Creating the Prototype

The prototype is established directly from the verb. The function DEFINEPROTO is used to create a property list entry for the verb consisting of a list of the case names to appear in the finished prototype

form. At present, prototypes for draw, erase, and name are determined in this way. They consist of exactly the same form, although this need not necessarily be the case. Other verbs, of course, appear in the input string, but these are reduced to the canonical verb form so that the reference to the prototype is always done correctly. The function PROTO is used in the ATN grammar to retrieve the proper prototype form.

As constituents are discovered and parsed by the ATN grammar, they are slotted into the proper case positions. The function PROTOTYPE changes the current prototype to reflect the value to be added. Normally, additions are made as they are discovered, but the HOLD mechanism allows some of these decisions to be delayed until a later VIR arc can find the proper positioning of the constituent. This is particularly useful in handling touches which are found out of place and for moving prepositional phrases to the post-verbial position.

3.5 The QA Interface

The Language Analyzer utilizes knowledge accumulated in KB to help in parsing and to answer user questions. The two functions, QACALL and QASK provide the interface to QA for grammar-produced and user-initiated questions respectively (see Figure LA-1)

QACALL builds a message from the arguments passed to it. The first argument indicates the name of the operation required of QA and this is limited to either FINDOBJ, CONVERT, FINDDRAW, or INFO. Other arguments to QACALL are included as part of the message to QA. Only FINDOBJ and CONVERT are used in grammar-produced questions, while user-initiated questions may use FINDDRAW and INFO.

QASK is activated by the LA Controller after the parser returns with an indication that it parsed a question form. This function must determine how to answer a user's question based on register settings created during the parse. The collection of registers is passed as the function's one argument. If the user's answer has not already been determined during the parse, then an appropriate message is sent to retrieve additional information from QA.

The function FINDOBJ is used in determining the reference of a name mentioned in the input string. For phrases like "the point P" it answers the questions "Is the object named P a point?" and "What is the node identifier in memory for the point named P?" The parser calls on this function whenever a name is mentioned. This is especially useful in

determining how a word is being used when no lexical entry can be found.

Conversions to system units are accomplished through the CONVERT feature of QA. The only units recognized in MG are screen units for lengths and degrees for angles. Any other scale must be converted to these and the grammar contains appropriate calls to QA to accomplish this.

FINDDRAW is used to determine if a particular entity is drawable or erasable, and to retrieve the list of objects drawable by the system. Its use is limited to answering specific questions like "Can you draw ..." or "What can you draw?"

The INFO feature of QA encompasses all other types of questions. A prototype structure, similar to that built as a final form of representation for completed sentences, is partially filled in with the information known at the time of the call and passed as a parameter. In addition, a list of the cases that need to be filled by INFO is also passed. The response arrives in the form of a prototype with the requested cases being filled if possible. If more than one item can fill a case slot, then all of these are returned. For example, in answering a question like "What did you draw here <T>?" the question is put to INFO in the form "Give me the %OBJECT and %NAME of the items drawn at <T>.". The response is later passed along to GEN so that the eventual answer comes out as, for example;

"a CIRCLE named BALL"
"a POINT named FRED"

3.6 Problems in LA

A number of minor problems continue to be a point of concern in the language analyzer. Of course, it is difficult to assess the ultimate limitations of the approach taken here, but hopefully no major obstacles will be encountered in the immediate future to impede the extendability of the system. A few problems are discussed below.

The adjectives "small" and "large" should be handled heuristically. In doing this, the object must be considered and appropriate size calculations must be made relative to its shape. The current approach is adequate as a guess for average size circles on the screen, but a "one inch diameter" small circle and a "four inch diameter" large circle may not always be appropriate. This problem is not unique to these two words. Consider, for example, how to handle "near" or other locatives such as "above", "below", "behind", or "next to".

Prepositional phrases containing "to" and "from" are handled identically. They each result in creating an ENDPOINT tag in the %POSIT case of the prototype. Thus, the directionality of the statement is lost. Such a distinction was deliberately not made since a problem can arise in the graphics routines if a line is drawn in one direction and an erasure is attempted in the other. Some simple solutions to the "endpoint" problem exist and should be forthcoming.

In parsing, the grammar should perform more consistency tests with objects and their parameters. Currently, the grammar allows input sentences to specify circles drawn at angles, lines drawn with circumferences, and points to be drawn "from here to there".

The prototype itself may not be easily extended to handle, for example, a "move" operation. One manner of specification in a move could indicate an origin and a destination position. This might require introducing sub-cases into the %POSIT case. Other new operations may require similar extensions.

The verb largely determines what structure gets built during parsing. There may be no easy way out of this, but to extend the system to handle some simple elliptical forms like "A circle here <T> please" one may want a more flexible scheme for determining the prototype.

4. THE LANGUAGE GENERATION MODULE

4.1 Function and Overall Design

The generation of language responses to the user in the system is performed by the Language Generation module (GEN). A small collection of templates serves as the basis for generation (see Appendix L). A module can request that a response be generated by specifying a template number (T0, T1, T2, ...) and an ordered list of substitutions to be made for the asterisks which appear in the template.

Besides this straightforward method, there is another way that responses can get generated in the system. Instead of indicating the template number and substitution list as the two arguments in messages sent to GEN, a message may contain the indicator GENANS1 or GENANS followed by a prototype. This indicates that either one or many answers are required from the prototype given. A generative ATN grammar processes these requests by parsing the prototype and producing a list of template numbers and substitution lists, one for each response. The grammar

at this stage of development is rather simple (see Appendix M) but shows promise for future work.

The techniques used in the GEN module work adequately for NLG as it now exists. It is hoped that as improvements evolve in other modules of the system, GEN will prove to be expandable as well.

5. THE KNOWLEDGE BASE MODULE

5.1 Function and Overall Design

This module accepts a message passed to it from some other module and performs one of its two major tasks depending on a marker in the first part of the argument slot of the message. Control is passed to the Memory and Graphics section (an MG marker) or to the Question Answering section (a QA marker). The components of the KB module are shown in Figure KB-1. The MENTAL functions [Shapiro 1974] occupy 1.5K words, the initial knowledge occupies 3.5K words, and the rest of the KB LISP code is 7K words. The module has the following functions:

i. to answer questions about the stored knowledge posed by other modules in the system,

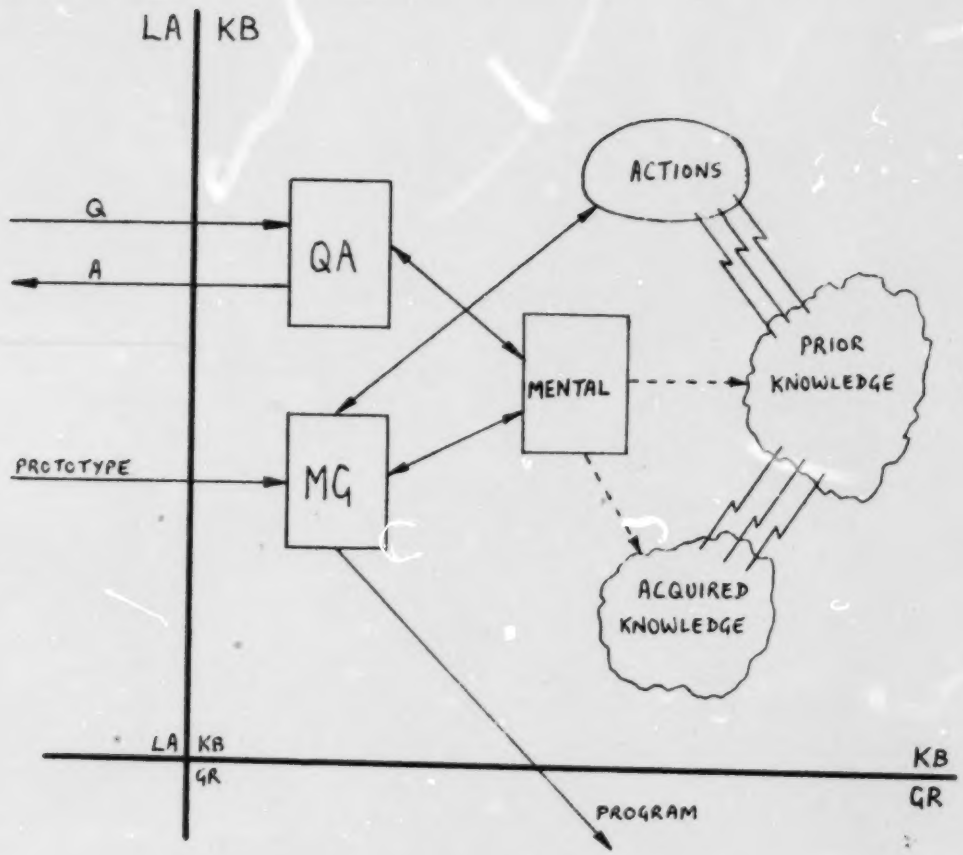
ii. to build, erase, and change semantic memory structures,

iii. to produce a message to be sent to the Graphics module which will control the drawing of an object on the screen.

5.2 The Semantic Memory

5.2.1 Structure

The style of network structures used is heavily influenced by those discussed in Brachman [1976]. They are manipulated and built using a slightly modified version of MENTAL [Shapiro 1974]. The network is used to encode both the system's prior knowledge about objects and details of the objects introduced during the discourse. Attached to the descriptions of objects in the network are actions (ACTs) which are encoded in LISP and used for checking structures, producing part of the output message, and providing information if it is not specified by the user. Appendix N gives a list of the actions and their functions, the Table below gives a list of the labels used on the Semantic Network links, while Appendix O describes the initial state of the network.



KEY:

Semantic Net

LISP code

LISP Code attached to Net

Manipulation

Attachment

Control flow

Figure KB-1

TABLE OF LABELS FOR SEMANTIC NETWORK LINKS

LABEL	STANDS FOR	COMMENT
SUBSETOF	subset of	connects two sets
ELMNTOF	Element of	connects item to a set
NAME	Has name	for item, or set, or concept
TYPE	Has type	for every node (SET, CONCEPT, ITEM, DESCRIPTION, ASSERTION, SYSTEM)
ELMNTDESCR	Element description	connects set to description of a typical element
DATTR	Description of attribute	from concept node to description
ROLE	Plays role	from description node
RESTR	Is restricted to	connects description to set
ACTION	to action	from description node to action name
ATTR	Has attribute	connects item to assertion
INSTNCOF	Is an instance of	connects item to concept
VAL	Has value	connects assertion to item or value
NUMBER	Number of occurrences	connects description to a number
STRUCTURAL	to a structural check	connects concept to an action
INSTANTIATES	Instantiates	connects assertion to description
OBJECT ERASED	to erased object	in simple screen model

The backbone of the network is a hierarchy of sets, including straight lines, points, angles, distances, defined points (i.e., locations on the screen; not necessarily illuminated), and circles (Figure KB-2).

Consider the portion of the network which stores prior information about circles (Figure KB-3).

Node 1 represents the set of all Circles. Node 2 represents the description of a typical element of that set. Node 3 represents one of the parts of the description. The part shown here is that of "Locator",

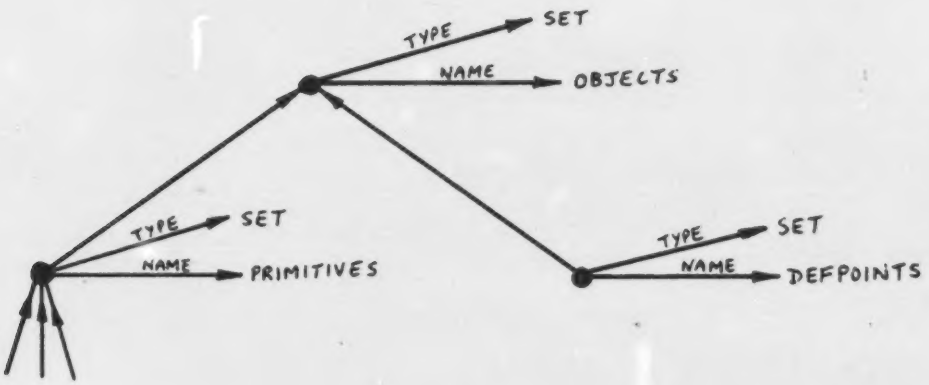


Figure KB-2

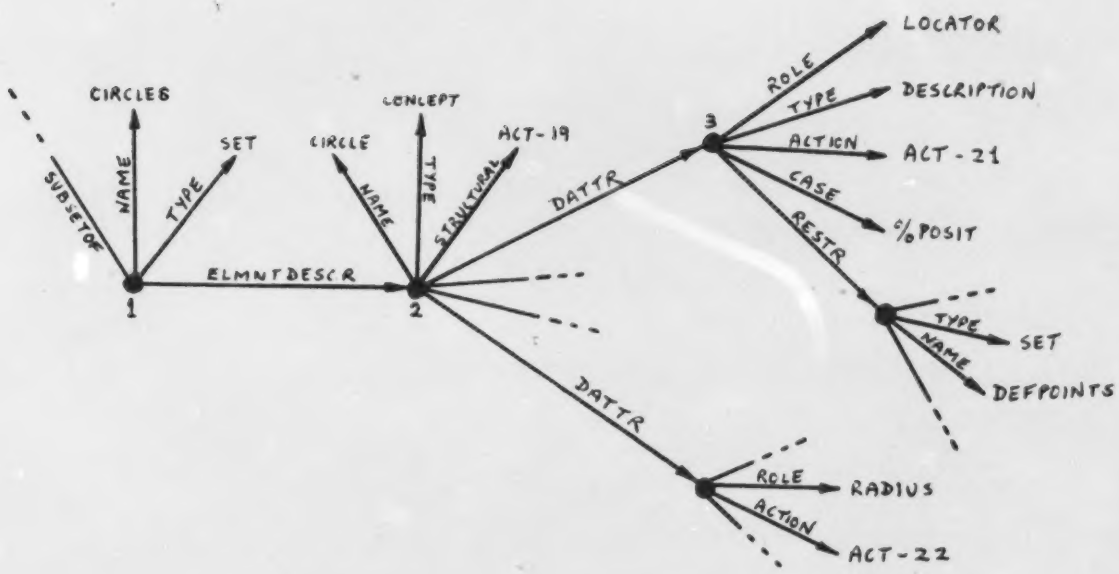


Figure KB-3

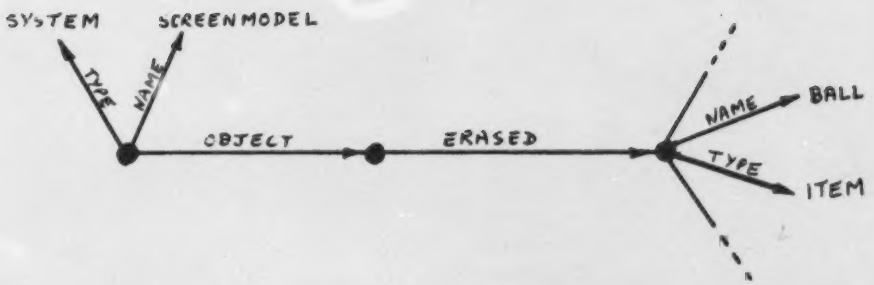


Figure KB-5

which is associated with the %POSIT case in the input prototype, and restricts things which play the role of locator to members of the set of defined points (Node 4). The ACTs will be explained later.

Figure KB-4 shows, as a result of a request to draw on the screen, how information about an object is stored in the network attached to the description of that type of object. Subparts of the object are also connected in this fashion.† Notice the "assertions" in the network, which include "the assertion (node 2) that the locator of the item (node 1) is a particular defined point (node 3)".

5.2.2 Actions

There are three types of actions attached to the network: the Structural action, the Default action, and the Todraw action.

The Structural act is used to check pieces of network after they have been built to the specification given in the input prototype. For example the net built for the new circle in Figure KB-4 is missing size information (i.e., Diameter, Radius, or Circumference). ACT-19 (see Figure KB-3) will detect this omission and will report that the role of Radius has not been instantiated. In general the structural act 'knows' what the minimum requirements are for drawing the object, and will return a list of those missing. In the case of a circle, the requirements are Locator and Radius.

The Default acts are activated if a non-null list is returned by a Structural act. In this case the act used is the one associated with the node which has a Role link to Radius (i.e., ACT-22; see Figure KB-3). The Default act will look to see whether there is anything in the network which will allow the missing value to be calculated (e.g., Radius calculated from Diameter), and if not, will provide a reasonable default value. When it has a value, it builds an additional piece of network and completes the requirements of the Structural act. A Default action is run for every omission. In the implemented system only circles have default actions. The Locator defaults to a touch point slightly to the right of the center of the screen, and the Radius defaults to 100 screen units (approximately 1.5 inches).

---

†For simplicity, many important links are not shown in Figure KB-4. Note that this is the form of the network prior to defaults.

"Draw a circle called Ball at (100,200)"

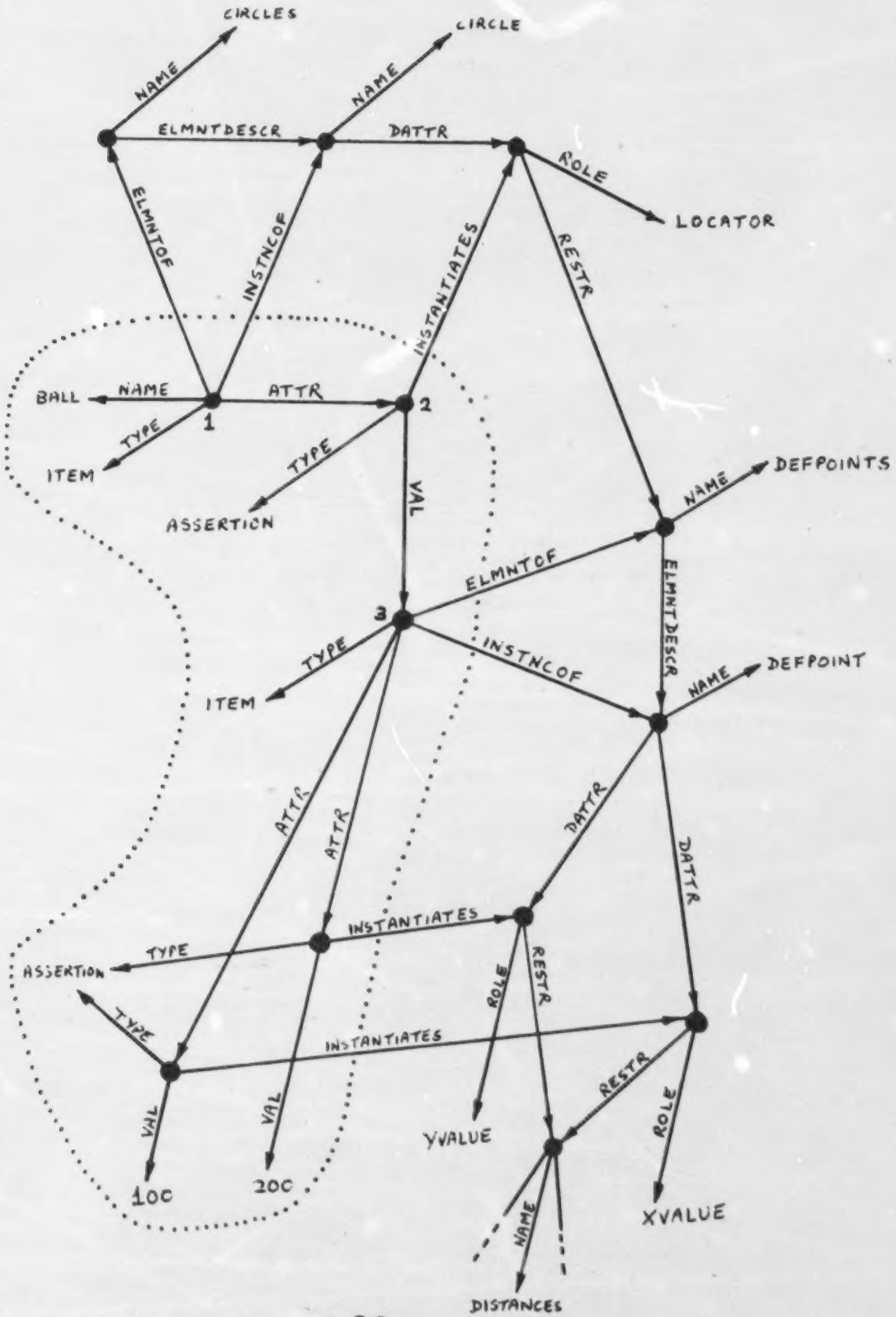


Figure KB-4

Once the item is fully specified the Todraw action can be used. It takes the piece of network representing the item to be drawn and produces from it a program form which will do the job. For example, the circle. with its center at the point (100, 200), and its defaulted radius of 100 units, will produce (F4CRCL 100 200 100). For a straight line the Todraw act is a little more complicated as it has a choice which depends on which combination of endpoint, midpoint, angle, and length has been specified. (See Appendix P).

5.2.3 Naming and Erasing

The user has the ability to ask for an object to be drawn, and name it, both in the same sentence (e.g., draw a circle called Ball at (100,200)). This results in a Name link from the item as in Figure KB-4. If a name is not specified then the system provides one by taking the digits from the MENTAL node identifier (e.g., N2345) and appending them to the first two letters of the concept name (e.g., CI2345). Every node generated by MENTAL is automatically given a unique identifier.

When the user requests that an object be erased from the screen the system leaves it in the network, but marks it as erased. This acts as a simple screen model, and prevents the system from attempting to erase an object that is already erased. (See Figure KB-5).

5.3 Question Answering

5.3.1 Introduction

This section of the Knowledge Base is activated when another module needs some information. The syntax of the various QA calls is given in Appendix Q. There are four available functions:

i. INFO - A general network searching function, and the most powerful function of the four.

ii. FINDOBJ - Used for discovering whether a word which is suspected of being the name of an object actually names some object on the screen.

iii. CONVERT - This is a function for converting lengths to screen units or angles into degrees.

iv. FINDDRAW - Used for finding out whether an object can be drawn.

5.3.2 The INFO Function

A prototype form and a list of keywords are passed as the two

parameters of INFO. The prototype form is used to search for items in the network. A list of items matching the description (i.e., the prototype is viewed as a description) is obtained and those items are further searched for the information requested by the list of keywords. For example:

```
(INFO (%OBJECT  %NAME  NODE  %POSIT)
      ( (%POSIT .( (LOCATOR  (100.200)) ) )
)     )
```

could give:

```
(     (%OBJECT . (CIRCLE POINT) )
      (%NAME   . (BALL   FRED)  )
      (NODE    . (N2345  N3442) )
      (%POSIT  . ( (100.200) )  )
)
```

i.e., a list of dotted pairs of the form (keyword . list). The answer to the question "Which objects are at (100,200)" therefore, is a circle named Ball and a point called FRED.

The INFO function is capable of handling positions specified by LOCATOR, ENDPOINT, and TOUCH. The latter will successfully match both LOCATOR and ENDPOINT during the search, but preference will be given to LOCATOR. Consequently, if the new inputs "What did you draw here <touch>?", and the touch point corresponds to the endpoint of a line and the center of a circle then the response will name just the circle. However, if the touch corresponds to two LOCATORS (or two ENDPOINTS) then both will be returned.

The function uses the ITEMFIND procedure which searches for all objects which match a given prototype. This is done by considering each of the cases in the prototype separately (e.g., %NAME, %SIZE), obtaining lists of matched items, and then forming the intersection of those lists. The resulting list gives the ITEM nodes of all objects which possess all of the required characteristics.

There are two error conditions for the INFO functions. If there is no object that corresponds to the description then the message "object as specified cannot be found" is output to the user. In this case, and in the case where the object is found but there is no information corresponding to the given keyword, the "list" returned with the Keyword is NIL, e.g., ((%POSIT . NIL)).

5.3.3 The FINDOBJ Function

There are two ways to use this function. The first is to specify both a suspected name and an object type in the call, e.g., (FINDOBJ BALL CIRCLE). This tests the network to see if there is an object of the specified type with the specified name. If there is, then the node identifier of that item is returned, e.g., ((NODE . N2345) CIRCLE) . If there is no object of that type with that name then NIL is returned, e.g., (NIL CIRCLE). If, by accident, an invalid object name has been given in the call, then NIL is returned instead, e.g., (NIL NIL). The second type of call allows the object type not to be specified in the call (i.e., NIL is used). This works in the same way as described above, except that on return it inserts the object type of the item with the name given in the call, e.g., (FINDOBJ BALL NIL) gives (((NODE . N2345) CIRCLE)).

5.3 The CONVERT Function

The response from this function is the value obtained after converting the given number into screen units, or degrees, depending on the unit specified, e.g., (CONVERT 3 INCH) gives (192). Note that the screen is 8 inches square, and 512 by 512 screen units.

5.3.5 The FINDDRAW Function

There are two uses for this function. The first allows the question "can you draw an X?" to be asked by using the call (FINDDRAW X). The response will be either T or NIL, for yes or no, respectively. The second use is to answer the question "what can you draw?", e.g., (FINDDRAW NIL) gives (ST-LINES POINTS CIRCLES). This function operates by searching the network for concepts with a TODRAW role. It is assumed that as they can be drawn, then they can also be named or erased.

5.4 Memory and Graphics

5.4.1 Introduction

On receipt of the message, the KB module examines the first item of the argument list. If it is "MG" the Memory and Graphics section takes control. The %TYPE value is examined next and control passes to the appropriate section depending on its value. Note that only imperatives (IMP) are implemented in MG. Next the %ACTION value is used to switch to the NAME, ERASE, or DRAW sections of KB.

5.4.2 Drawing

After finding the set and concept nodes for the object to be drawn, an item node is inserted into the network for this new object, and links are formed between it and the set and concept nodes. If a name is given (e.g., "Draw a circle called Ball") this is added to the item node, otherwise a name is generated by the system as previously described. The orientation, position, and size information is extracted in turn from the prototype, and appropriate ATTR, VAL and INSTANTIATES links are set up. Note that the positional information may include a (NODE . <node identifier>) form, due to sentences such as "connect P with (100,200) where P is the name of a point, and the node identifier is that of the item node for the point.

The Structural act is used to check the structure, as previously described, and if necessary, default values are inserted in the network. Once the net for the item is properly specified, the TODRAW action is used to produce a program form for inclusion in a message which is then sent to the GR module. This message contains a modified form of the prototype that was sent to the KB module (Appendix P). After the drawing is completed, the screen model is adjusted, and a message is sent to the INIT module to indicate successful completion.

5.4.3 Erasing

If the screen is to be erased, then the screen model is altered to mark all items as erased, and a message is sent to the GR module to erase the screen. On return from GR, the KB module itself sends a successful completion message. If an object is to be erased, and if that object has not been directly specified, then the network is searched for all items that match the prototype[†]. If more than one is found the items are checked and all those which have already been erased by the user are deleted from the list. From the items remaining, one is chosen to be erased. If the object has been specified then the form (NODE . <node identifier>) will be used. This removes the necessity for any further searching, and that object can be erased directly. The TODRAW act is used to produce a program

[†]Note that the positional information may include TOUCH forms. These will match both ENDPOINT and LOCATOR during the search.

form which describes the erasure. This is included in a message which is sent to GR. The only difference between a draw and erase message for the same object is that the %ACTION is marked as DRAW or ERASE, respectively. Consequently the program form can be produced by exactly the same Todraw action for both operations. The GR module detects the difference in the prototype and acts accordingly. After the erasing is completed, the screen model is adjusted to mark that item as erased, and a message is sent to the INIT module to indicate successful completion.

5.4.4 Naming

Note that "naming" in this system refers to the naming of objects drawn on the screen, and does not mean the naming of groups of objects (e.g., four straight lines being called a square). The first action is to remove the (%NAME . <name>) pair from the prototype, and then use the new prototype to search for all items that match that description. Here, as before, the positional information may include TOUCH forms. Those that already have user given names are discarded, and one item is picked from the resulting list. That item node is deleted from the net and a new node is inserted having the same links, except for the name link which points to the new name. Finally a message is sent to the INIT module to indicate successful completion.

5.5 Problems and Extensions

5.5.1 Moving

There are three types of operations of this type that we would consider adding to the system.

i. Draw in the same place with no change after a prior erasure.
e.g., "Draw L1." (a null move)

ii. Draw in a different place with no erasure of old object.
e.g., "Copy the circle here <touch>."

iii. Draw in a different place with erasure of the old object.
e.g., "Move L1 to here <touch>."

It would be nice if the mechanism which handles these could also provide changes in the values of other cases on request (e.g., "Draw L1 here but at 45 degrees and an inch smaller"). Many of these details have been worked out, but have not been implemented. One reason for not implementing this was our suspicion that in the worst cases the prototype might need slight revisions. To do the above operations at all requires special

interpretation of the prototype in order to be able to specify the existing object being referred to, as well as the new requirements.

5.5.2 Remembering

We considered implementing an action that would allow the user to 'remember' the objects that he had drawn by having all or part of the semantic net dumped to a file. A 'recall' function could be used to read back information from specified files (i.e., those created by the remember function) and re-establish it as the semantic net. This pair of actions could be used to maintain a record of the state of an interaction, and to save complete or partially completed drawings between sessions. Remembered pictures could also be displayed, after appropriate preprocessing, on other devices such as a plotter or a CRT. In addition, it may be useful to have a 'forget' function which could be used to permanently erase net structures representing all, or part, of some item or items. This could be used if a mistake had been made, or if some object was no longer required. It could be argued that the erase action should remove the item from the network. However, this would remove the possibility of redrawing a named object after it had been erased.

5.5.3 Defaults

A simple extension of the default mechanism would be to mark all pieces of the net which were built as a result of default acts. This would allow the system to recognize the difference, and enable the system to remind the user of exactly what he had specified for an object. In addition, defaults could be provided for all of the primitives, whereas at present, only circles have default actions. The only additions required would be to slot the default and structural acts into the code for KB, and create links from the appropriate nodes of the concepts to the act names. The activation mechanism will work for all defaults in the same way.

Structural acts return a list of those roles which need to be instantiated before the Todraw act can function correctly. It may be the case that the Default actions should be executed in a preferred order so as to make maximum use of the information given and to minimize the number of defaults.

For a better treatment of defaults, two extra techniques would have to be used. The present implementation of defaults is essentially context-free, and merely uses built-in values. There will be situations when this

is not sufficient, and where a heuristic default mechanism would be appropriate. For example, a default action could execute a space-finding routine to find a reasonable location for the object on the screen. In other situations it would be sensible to ask the user for the extra information, if, for example, the heuristic fails to find a suitable value.

5.5.4 Structural Information

There are strong grounds for arguing that the Structural act contains too much information, even for such simple items as point, line, and circle. This information, for example what fully defines a straight line, is not accessible as it is encoded in the LISP code. For even only slightly more complex objects, such as square, it appears that this information should be encoded in a network form. This would allow a new use of QA by LA. During parsing, to aid in establishing the role of prepositions, it would be useful to have LA ask QA what other cases to input, given the current input. For example, having found an angle and endpoint, and something else which is not easily parsed, QA would be able to suggest trying to look for a length description. A structural network would play the important role of specifying the relationships between the various attribute descriptions.

5.5.5 Defined Points

At present, whenever a defined point is specified, no attempt is made to determine whether there is already a structure for that point. A simple extension of KB would be to include a routine to check for existing structures. This would mean that lines which shared the same endpoint would share the same Defpoint structure.

5.5.6 Point Notation

It might be possible to use a unified notation for points, lines, and areas. For example:

a particular point	<100, 200>
the line y = 200	<vx, 200>
any point, with y = 200	<? , 200>
any point	<? , ?>
the screen	<vx, vy>
a horizontal line segment	<50 - 70, 200>
a rectangle	<50 - 70, 100 - 150>
any rectangle	<x1 - x2, y1 - y2>

In a more sophisticated system it might be useful to have volumes as primitives and other basic objects as degraded volumes.

5.5.7 Screen Model and Action History

In order to handle words such as inside and outside (e.g., "please draw a circle inside the triangle") it is necessary for KB to have good knowledge of what is on the screen. For 'outside', for example, it is necessary to find space for the object to be drawn. The stored knowledge about items in the semantic net in some way provides this information, but space-finding could involve inspecting every item, and much calculation. Consequently we would prefer to have either special hardware to tell KB which lines or points are illuminated, or some sort of comprehensive model of the screen, possibly in a bit-map form. Such a model could be used for space-finding, and discovering spatial relationships and topological properties.

To deal with questions of the kind "What did you do?" and "Why did you do?", an action history should be included in the system, logically in the KB module. This would consist of a time ordered set of actions with their associated details. Use could be made of the simple indexing mechanism available through set membership links. These would point to the set of all objects of the same type, and they in turn would be linked via a "time line". This sort of mechanism could also be used to help with anaphoric reference.

5.5.8 Functions

For positional indications such as "near", "at", or "center" we will probably need to have words that trigger functions in the KB module. This has not been carefully worked out yet, but we feel that forms such as (%POSIT . (NEAR (TOP #SCREEN))) or (LEFT (UPPER #SCREEN)) could be used. In line with our general principle that as much information as possible should be extracted from a sentence before KB gets to act on it, we feel that QA would probably be the correct place to implement such function handling. This would allow LA to pass areas or coordinates to the MG section.

6. THE GRAPHICS MODULE

6.1 Introduction

This module accepts messages containing screen alteration commands and executes them. First the message is inspected to see what the action is, and a marker is set depending on whether it is draw or erase. Then the argument part of the message is executed one step at a time. In the current system only one step is present in each message (see Appendix P

for GR message format). The marker sets the underlying graphics machine code routines into either draw or erase mode. An object is erased by drawing over it in erase mode.

The development of a LISP/Plasma panel package (PLASUB) interface enabled us to program the module entirely in LISP. The interface also allowed the use of trigonometric routines from the FORTRAN Library, and consequently we were able to do the angular calculations with ease. The reason for the F4 prefix on the drawing function names is that at one time these were to be written in FORTRAN.

6.2 The Screen Functions

The draw/erase functions consist of point drawing, circle drawing, screen erase, and line drawing routines. There are four line drawing routines, one for each sensible combination of the two endpoints, the midpoint, an angle, or a length. The point drawing routine uses the increment mode of drawing to draw a point consisting of four dots. The line routine uses a basic vector operation. The circle routine uses short chord vectors to produce the circle, with the angle step size that defines the chord varying depending on the size of the circle required.

The PLASUB package routines used in GR are:

ERASE - screen erase
SETMOD - set mode of panel
INCRE - incremental draw/erase
VECTOR - linear draw/erase/move

Working with the plasma panel involved us with three different coordinate systems.

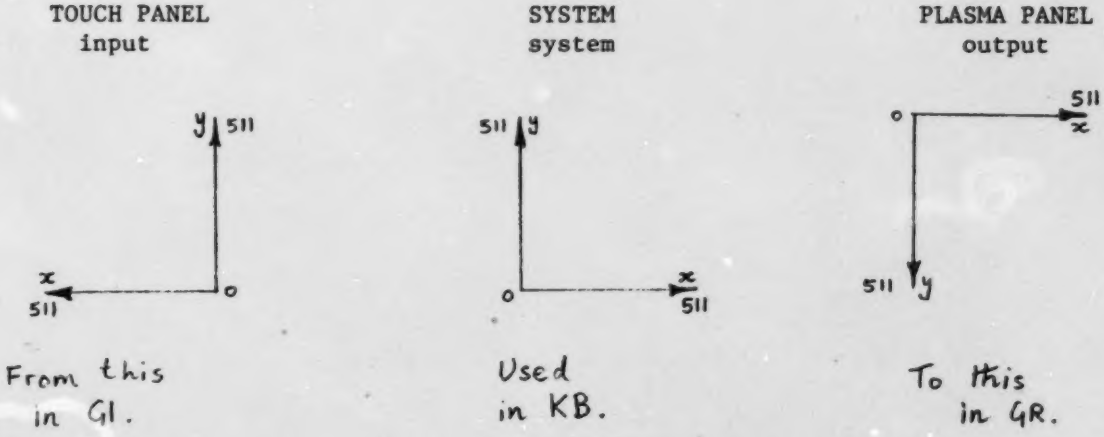


FIGURE GR-1

6.3 Problems and Extensions

The circle routine is less than perfect at the moment, but the authors are aware that there are better circle drawing techniques that could be used if necessary. Because of the large number of vectors used in drawing a circle, and because of their frequency, the circle drawing routine suffers occasionally from transmission errors, as the panel is connected remotely to the DECSystem 10. It may be that using increment mode for circles would improve this.

7. CONCLUSION

7.1 System Performance

7.1.1 General

The system responds fairly quickly to user input, with the only delay being due to the fact that the terminal is remote from the computer and not on the fastest possible line. In consequence there is a substantial wait while circles are being drawn, aggrevated by a slightly inefficient circle drawing routine and by being swapped-out occasionally. The sentence processing however, runs in approximately real time, with an average processing time of well under a second from input to prototype formation. A slight speed-up of the system could be obtained by compiling the KB module of the system. This was not compiled due to the fact that a large number of structural changes would have had to be made before compilation could take place. The average processing time for each sentence from input to completion is about 2 seconds, and depends on sentence type, sentence length, and screen operation. The number of sentences that can be put on the screen is about fifteen, and the number of KB-structure-building sentences (drawing commands) that can be processed without running out of space is about ten. It should be noted that as a spinoff from this project we now have a LISP-FORTRAN interface which may be used for other LISP projects involving graphics. A version of LOGO has already been programmed using this interface.

7.1.2 Good Things

The technique of modular construction using message passing and a control executive proved to be very successful. We were able to develop well defined interfaces, and pursue development of modules individually using module testing routines. Modules were initially built as "dummies" and gradually built up to work correctly. We were able to use dummy modules to build a complete system including, but isolating, a newly changed module. The ability to monitor the messages in the executive allowed us to diagnose interfacing problems as well as errors within modules.

We were pleased with the relative ease with which it was possible to construct the system, given the use of the ATN and MENTAL packages, and with the knowledge of a "handful" of AI techniques. We take this to be an indication of the progress of AI technology.

Despite the limited grammar, and the relatively simple defaults and system heuristics, interacting with the system was pleasant, and its "intelligence" was very convincing. Several people who had had some Computer Graphics experience reacted favorably to demonstrations of the system, and were inclined (as even we were sometimes) to ascribe the system with greater intelligence than it was capable of displaying.

The ability to touch the screen with one's finger proved to be a useful addition to the stilted forms of language normally investigated in current research on Natural Language Understanding Systems. The processing of deictic expressions, including place adverbs and demonstratives, followed directly once touches were included..

The use of the INFO function in the QA section of KB allowed the user to specify objects by using a partial description. We feel that this is a powerful technique, and contributes a great deal to the ease of interaction with the system.

7.1.3 Bad Things

An overriding problem with the system was that, despite our best efforts, it tended to grow on its own. Some unplanned-for problems inevitably occurred, and were solved, but due to constraints on time and effort were not necessarily solved in the best way. As usual, there are bits we would like to rewrite.

It is not clear whether the simplistic world of lines, points, and circles truly provides a basis for assertions about more complicated worlds. In addition, we were a little suprised that even with just lines, points, and circles, and drawing and erasing, the system had to deal with quite complicated situations. This may be partly a reflection of our own naiveté, and partly a strong indication that even the simplest of AI systems should not be underestimated.

It would be difficult for us to claim that our work resulted in a practical system. From a hardware point of view, a major weakness is that the touch panel does not allow very fine resolution, and consequently, precise placement of objects must be done using coordinates. The system also lacks many of the things that one would expect in a Computer Graphics system, however, this is to be expected as it was never our intention to implement all of those facilities in the pilot project.

More extensive modes of graphical input would certainly enhance the communication in a graphics environment. Specifically, one often would like to simply sketch out an object rather than verbally describe it. Development of a representation which would model the essential characteristics of sketches would greatly aid such an effort.

The physical management of the screen needs some improvement. Since the user is permitted to draw anywhere on the screen, text and pictures often overlap. Providing a division in the screen between the drawing area and the area for text seems to be the best solution. The text area could be at the bottom of the screen showing only the last input and response.

The limited syntax of the pilot system is too prohibitive. A response of "please rephrase" doesn't provide the user with many clues as to what was wrong with his input [Weischedel 1977]. Certainly, an intelligent system would at least be able to say that it had no knowledge of squares if asked to draw one. Likewise, if the input contained all of the essential parts to specify drawing an object, but the grammar didn't permit the form, the system should override the grammar.

We felt that as a tool for manipulating knowledge MENTAL was at too low a level. In addition to this, the Brachman-like formalism adopted for KB uses fairly complex structures. This meant that a simple insert or change involved several operations. If we could start again, we would build a higher level of language over MENTAL and use that for net manipulation [Bobrow 1976]. We are aware that MENTAL has evolved, and that newer versions may be significantly easier to work with [Shapiro 1976].

7.2 Outstanding Problems

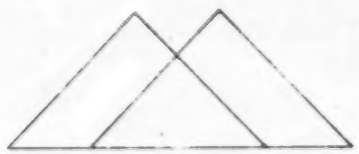
Language generation, as it exists in the pilot NLG system, is quite weak. Although the foundation has been laid for developing this module, little effort has been given to this. Nevertheless, some very simple mechanisms have proved quite effective in allowing an adequate level of communication. Language is generated under the guidance of an ATN generative network which suggests a template and a list of substitutions. Thus, output is limited in form to prestored templates. This requires that all system responses be anticipated to some extent when the templates are designed. Also, the knowledge base

remains inactive during generation, unlike in language analysis. GEN should be able to use the QA interface to obtain additional information to use during generation, and it should also have access to the lexicon used by LA.

There are a number of interesting but outstanding problems connected with the KB module. The first has already been alluded to slightly: that of developing a full action history and model of the screen. The ability to refer to past actions and their ordering in time would allow a whole new range of questions to be asked. It would also help with reference problems, as in, for example, "Now connect the other ends", or "Draw another". The screen model, if it can in fact be built, would allow heuristic placing of objects and use of heuristics such as those for "near", or "beside" without undue calculation.

We believe that if knowledge is to be collected together at all in a system then as much knowledge as possible should be included so that rich interconnections may be established. It is reasonable therefore, to include lexical information in KB along with descriptions. For example, the word "CIRCLE" and its associated markers could be stored with the concept of circle in some way, just as a print-name can be stored on a property-list. There is no reason why the alphabet shouldn't have its set of concepts, so that the words attached to concepts would be made up of ordered sets of items which were instances of letter concepts. Thus letters could be drawn on the screen using the same general procedure as for objects. Fahlman [1975] makes a similar point.

A problem that would have to be faced if the system were expanded is that of erasing parts of objects. There are at least two subproblems. Consider the figure produced by drawing two overlapping triangles:



We will presume that triangle has been defined for the system. If the user requests that parts of lines which are inside another triangle be erased, the result should be:

This would involve some geometric calculation, and would leave parts of defined parts of objects to be represented. This may be a representational problem. Another problem occurs if the user requests that the inside triangle be erased:

The problem here is that the system does not "know" that the overlap of the two triangles is itself a triangle. A similar situation can occur if several objects are put together so that the spaces between them form some recognizable shape.

The problem of object selection by touching has been solved rather simply in the present system by selecting the object if the touch coincides with its defined location point, or, alternatively, some other defined point. However, in a realistic system one would want to allow touches to other places in order to select the object. For example, a circle could be selected by touching some point on its circumference, instead of its center. It should also be possible to select some closed figure by touching some point which lies inside the figure. A complete theory of object selection is a hard and extremely interesting problem, and would be a necessary addition to any practical system.

Of the problems presented in section 5, but not discussed here, the most challenging are those of heuristic defaults and the network specification of structural conditions. Brachman [1977] has more to say about the latter problem.

7.3 The Future

This section describes some of the topics in which the members of our group are carrying out research. We are now using the NLG pilot system as a background for new ideas and will be concentrating on the three areas of language analysis, knowledge representation, and protocol analysis.

Language analysis in a practical system should be robust. A practical habitable system should possess the ability to respond intelligently to sentences regardless of the completeness of the grammar. Since a complete grammar of English in this practical sense does not exist, research in natural language understanding must therefore develop methods that account for all potential input. Under investigation is the development of a language analyzer which combines syntactic with semantic processing forming a uniform model which approaches this capability. It is believed that several additional benefits will accrue from this approach. This scheme should be capable of correctly processing many elliptical and some ungrammatical sentences. In addition, semantic cohesion should override poor syntactical form. This would allow frequent users to invent shorthand forms conveniently with no alterations to the language processor. For example, in an NLG system one could use (17) and (18) instead of longer forms.

(17) DRAW CIRCLE (200,300)

(18) LINE <touch> <touch>

Work in knowledge engineering will attempt to improve and extend the representation used in the pilot system in order to provide a knowledge base which uses a homogeneous representation of many different kinds of information. It should, for example, include the structure of an object, how to draw it, the object's function, and relationships among drawn objects. The knowledge base should serve as a lexicon, a database of facts, and a graphical database. In addition, a portion of the research will be concerned with conversion between the different types of information, either by spontaneous computation or on demand.

Protocol analysis should lead us to discover adequate vocabulary and grammar and preferred modes of man-machine interaction, particularly ratios of graphical to language use and usage patterns. Some information about semantics can also be gathered by recording user reactions to system responses.

In addition to the effort in the topics already discussed, we are concerned with system organization, language generation, graphics, and processing drawn input.

7.4 Summary

This report has described a successful experimental program for manipulating simple pictures on a computer graphics terminal. Natural language and touch input can be used to give commands to draw, erase and name, and to pose questions about the objects on the screen and the capabilities of the system.

We are confident that with a robust and powerful language analyzer, mixtures of language and pictures for both input and output, stored knowledge about the pictures being manipulated, and an inference capability to assist the user, we will have a productive research and applications tool. Natural Language Graphics provides a fertile area for significant research and larger, more general NLG systems should result from our work.

8. REFERENCES

Agin, G.J. (1972) "Representation and Description of Curved Objects", Memo AIM-173, Standford AI Project.

Badler, N.I.(February, 1975) "Temporal Scene Analysis: Conceptual Descriptions of Object Movements", TR-80, Department of Computer Science, University of Toronto.

Bobrow, D.G., & Winograd, T. (November, 1976) "An Overview of KRL, A Knowledge Representation Language", Report No. STAN-CS-76-581, Computer Science Department, Stanford University.

Brachman, R.J. (1977) "The Evolution of a Structural Paradigm for Representing Knowledge", Ph.D. Dissertation, Harvard University.

Brachman, R.J. (October, 1976) "What's in a Concept: Structural Foundations for Semantic Networks", BBN Report No. 3433.

Coles, L.S. (1968) "An On-Line Question-Answering System with Natural Language and Pictorial Input", Proceedings of 23rd ACM National Conference.

Fahlman, S.E. (May, 1975) "Thesis Progress Report: A System for Representing and Using Real-World Knowledge", MIT AI Laboratory, Memo 331.

Kirsch, R.A. (August, 1964) "Computer Interpretation of English Text and Picture Patterns", IEEE Transactions on Electronic Computing.

Minsky, M. (1974) "A Framework for Representing Knowledge", AI Memo 306, MIT AI Laboratory.

Nevatia, R. (1974) "Structured Descriptions of Complex Curved Objects for Recognition and Visual Memory", Memo AIM-250, Stanford AI Project.

Palmer, S.E. (1975) "Visual Perception and World Knowledge", in: Rumelhart and Norman (eds.), Explorations in Cognition, W.H. Freeman Press.

Quam, L.H., & Diffie, W., "Stanford LISP 1.6 Manual", SAIL Operating Note 28.7.

Shapiro, S.C. (December, 1976) "An Introduction to SNePS", Technical Report No. 31 (revised), Indiana University, Bloomington, Indiana.

Shapiro, S.C. (May, 1974) "IU - MENTAL, A Working Paper", Computer Science Department, Indiana University, Bloomington, Indiana.

Simmons, R.F., & Bennet-Novak, G. (1975) "Semantically Analyzing an English Subset for the Clowns Microworld", American Journal of Computational Linguistics, microfiche 18.

Sondheimer, N.K. (1976) "Spatial Reference and Natural Language Machine Control", International Journal of Man-Machine Studies, Vol. 8.

Weischedel, R.M. (February, 1977) "Please Re-Phrase", Technical Report No. 77/1, Department of Statistics and Computer Science, University of Delaware.

Winograd, T. (1972) Understanding Natural Language, Academic Press.

Woods, W.A., Kaplan, R.M., & Nash-Webber, B. (June, 1972) "The Lunar Sciences Natural Language Information System: Final Report", BBN Report No. 2378.

Woods, W.A. (1973) "An Experimental Parsing System for Transition Network Grammars", in: R. Rustin (ed.) Natural Language Processing, Algorithmics Press.

9. APPENDICES

A. Module Sizes
B. Syntax of ATN Grammar Specification
C. Input Vocabulary
D. Partial Lexicon
E. Sentence-Level Grammar
F. Touch Grammar
G. Noun Phrase Grammar
H. Prepositional Phrase Grammar
I. Question Grammar
J. Sample Sentences and their Parse Times
K. Description of the Prototype
L. Language Generation Templates
M. Generative ATN Grammar
N. List of Actions used in Semantic Network
O. KBNET - Initial State of KB Network
P. The GR Module Interface
Q. Syntax of Calls to the QA Section of KB Module

		Words (Decimal)
ERR		26
EXEC		158
GEN		
	Templates	80
	ATN Network	359
	LISP Code	199
GI		23
GR		297
IN		233
INIT		112
KB		
	MENTAL Functions	1153
	Initial Network	3500
	LISP Code	7000
LA		
	ATN Network	5668
	Lexicon	1927
	LISP Code	2017
Misc. Functions		517

Appendix A: Module Sizes

1. <atn net> ::= (<arc set>$_1$)
2. <arc set> ::= (<state><arc>$_0$)
3. <state> ::= any ATN state name
4. <arc> ::= (CAT <category> <test> <action>$_0$ <term act>) or
   (JUMP <state> <test> <action>$_0$) or
   (MEM <words> <test> <action>$_0$ <term act>) or
   (POP <form> <test>) or
   (PUSH <state> <test> <preact>$_0$ <action>$_0$ <term act>) or
   (TST <label> <test> <action>$_0$ <term act>) or
   (VIR <constit> <test> <action>$_0$ <term act>) or
   (WRD <words> <test> <action>$_0$ <term act>)
5. <action> ::= <storage act> or <retrieve act> or <create act>
6. <storage act> ::= (SETR <reg> <form>) or
   (SETRQ <reg> <expression>) or
   (LIFTR <reg> <form>) or
   (LIFTRQ <reg> <expression>) or
   (HOLD <constit> <form>) or
   (PROTOTYPE <proto case> <form>) or
   <form>
7. <retrieve act> ::= (GETR <reg>) or
   (GETF <feature>) or
   (RFEAT <feature> <form>) or
   (CTGY <category>) or
   (NULLR <reg>) or
   (NEXTWORD) or
   (PROTO <reg>) or
   (QACALL <qafunct> <qa args>) or
   FEATURES or
   * or
   <form>
8. <create act> ::= (BUILDQ <fragment> <reg>$_0$) or
   (LIST <form>$_0$) or
   (APPEND <form> <form>) or
   (QUOTE <expression>) or
   (UNIONP <proto> <proto>) or
   <form>
9. <category> ::= (<ctgy>$_0$) or <ctgy>
10. <ctgy> ::= any lexical category
11. <test> ::= <action> or T
12. <term act> ::= (TO <state>)
13. <words> ::= (<word>$_0$) or <word>
14. <word> ::= any lexical word
15. <preact> ::= (SENDR <reg> <form>) or
    (SENDRQ <reg> <expression>) or
    (! <form>)
16. <label> ::= a LISP atom
17. <constit> ::= a LISP atom
18. <reg> ::= any ATN register name
19. <form> ::= <action> or any LISP expression to be EVALuated
20. <expression> ::= any LISP expression
21. <proto case> ::= a case name in the prototype
22. <feature> ::= any lexical feature
23. <qafunct> ::= FINDOBJ or CONVERT
24. <qa args> ::= argument list for QA
25. <fragment> ::= a LISP skeletal expression using * and + for substitutions
26. <proto> ::= an NLG prototype

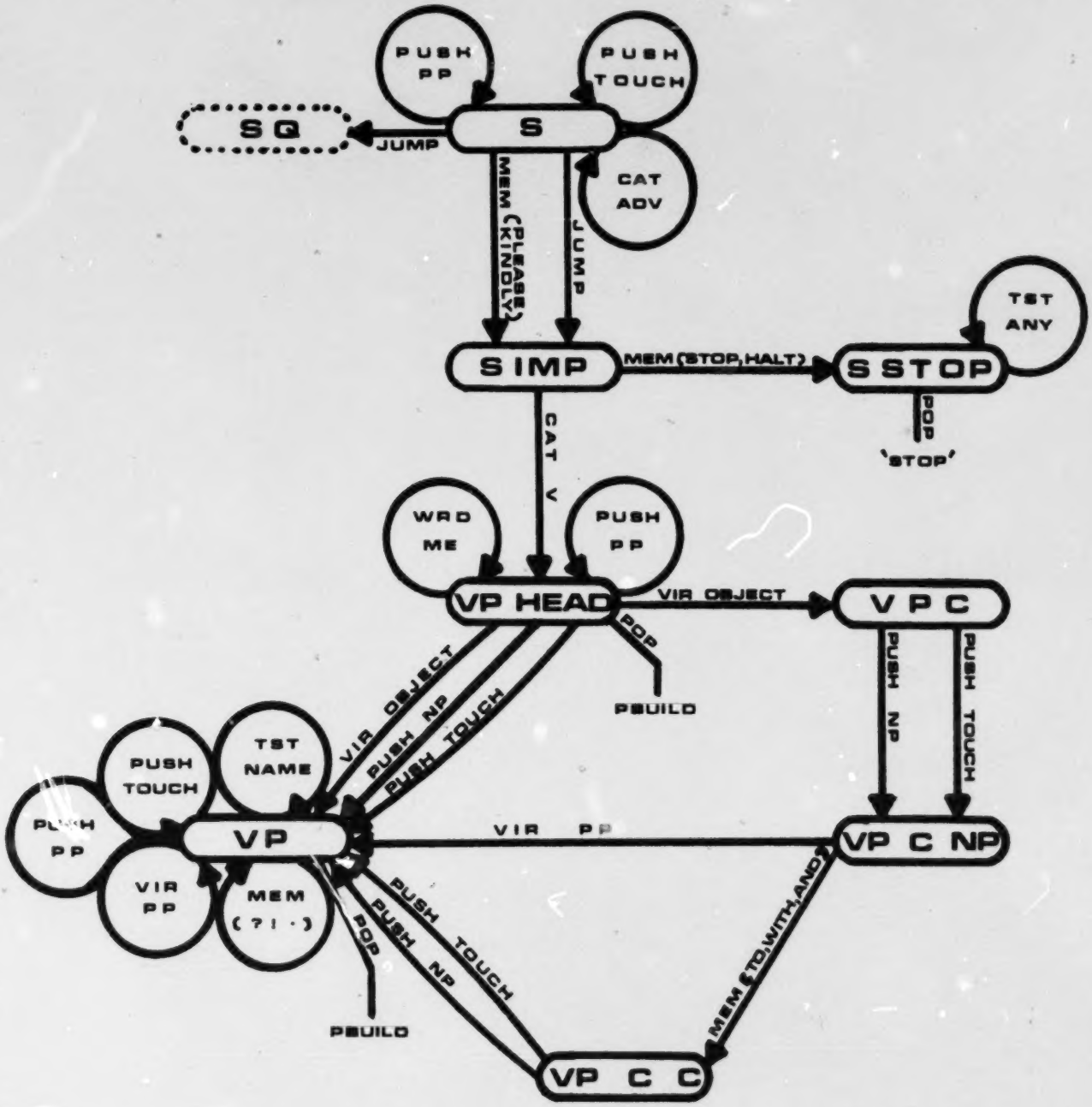
Appendix B: Syntax of ATN Grammar Specifications

A
AN
AND
ANGLE
ARE
AT
BETWEEN
CALL
CALLED
CAN
CENTIMETER
CENTIMETERS
CENTIMETRE
CENTIMETRES
CIRCLE
CIRCLES
CIRCUMFERENCE
CM
CONNECT
CONSTRUCT
COULD
DEGREE
DEGREES
DIAMETER
DID
DO
DOES
DRAW
EIGHT
EIGHTY
ENDPOINT
ENDPOINTS
ERASE
FIFTY
FIRST
FIVE
FORTY
FOUR
FROM
HALT
HAS
HAVE
HERE
HORIZONTAL
HOW
HUNDRED
INCH
INCHES
IS
JOIN
KINDLY
LARGE
LENGTH
LINE
LINES
LONG
MAKE
MANY
ME
MILLIMETER
MILLIMETERS
MILLIMETRE
MILLIMETRES
MM
NAME
NAMED
NAMES
NINE
NINETY
NOW
OBJECT
OBJECTS
OF
ONE
OR
PLEASE
POINT
POINTS
PUT
RADIAN
RADIANS
RADIUS
SCREEN
SEVEN
SEVENTY
SIX
SIXTY
SMALL
STOP
STRAIGHT
TEN
THAT
THE
THEN
THERE
THING
THINGS
THIRTY
THIS
THREE
THROUGH
TO
TWENTY
TWO
UNIT
UNITS
VERTICAL
WHAT
WHERE
WHICH
WITH

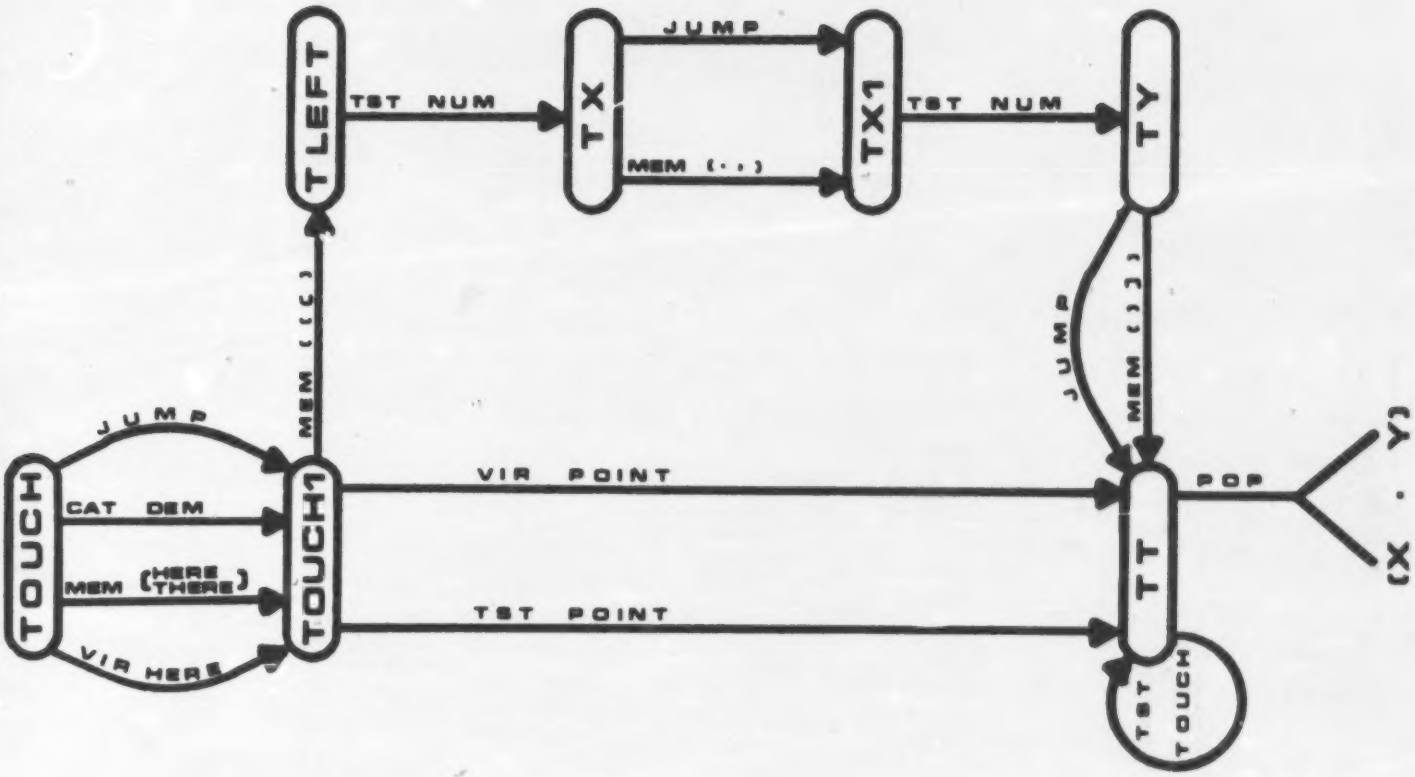
Appendix C: Input Vocabulary

WORD	CTGY	ROOT	NUM	TRANS	CASE	TAG	OBJ	%OBJECT	SIZEOF	ADJ	DEF	ABBREV
A	DET	-									NIL	
AND	CONJ	-										
ANGLE	N	-			(%ORIENT)							
ARE	COPULA	IS	PLUR									
AT	PREP	-			(%ORIENT %POSIT)	LOCATOR						
BETWEEN	PREPC	-			(%POSIT)	ENDPOINT						
CALL	V	NAME		T								
CALLED	COMPL	NAME			(%NAME)							
CAN	AUX	-										
CIRCLE	N	-			(%OBJECT)							
CM	N	-	PLUR		(%SIZE)				LINE			T
CONNECT	V	DRAW		T			ST-LINE					
DEGREES	N	DEGREE	PLUR		(%ORIENT				ANGLE			
DRAW	V	-		T								
EIGHT	NUM	8										
ERASE	V	-		T								
FIRST	ORD	-										
FROM	PREP	-			(%POSIT)	ENDPOINT						
HERE	ADV	-			(%POSIT)							
HOW	QADV	-										
INCH	N	-			(%SIZE)				LINE			
LARGE	ADJ	256			(%SIZE)							
LINE	N	ST-LINE			(%OBJECT)					(STRAIGHT)		
LINE	N	CURVE			(%OBJECT)					(CURVED)		
ME	PRON	-										
OF	PREPF	-			(%SIZE %ORIENT)							
PLEASE	ADV	-										
POINT	N	-			(%OBJECT)							
RADIUS	N	-			(%SIZE)			(CIRCLE)				
SCREEN	N	-			(%OBJECT)							
SMALL	ADJ	64			(%SIZE)							
THAT	DEM	-			(%POSIT)							
THROUGH	PREP	-			(%POSIT)	LOCATOR						
VERTICAL	ADJ	90			(%ORIENT)							
WHAT	QWORD	-			(%OBJECT %NAME)							
WHERE	QWORD	-			(%POSIT)							
WHICH	QDET	-										
WITH	PREP	-			(%SIZE %ORIENT %POSIT)	ENDPOINT						

Appendix D: Partial Lexicon

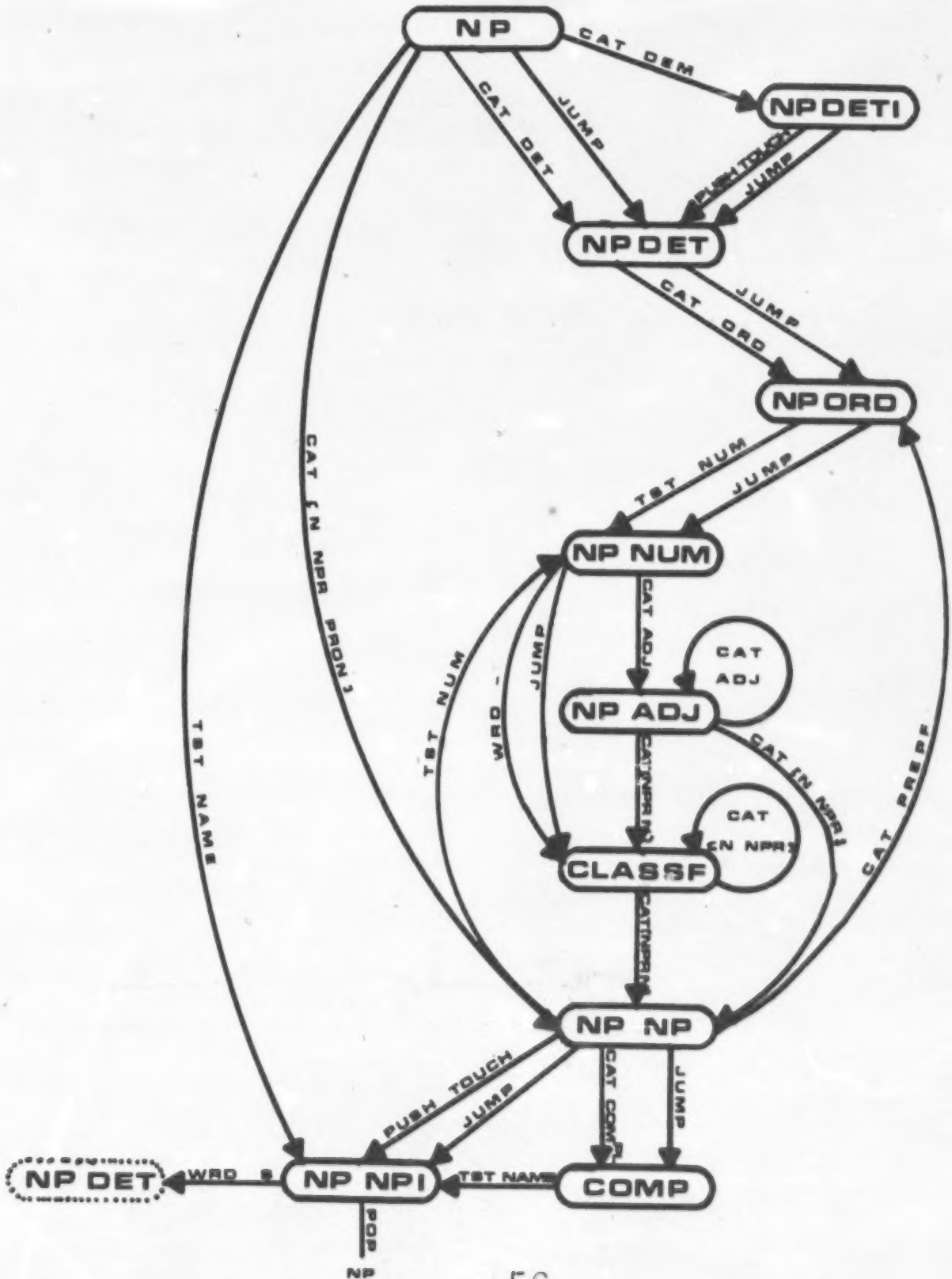


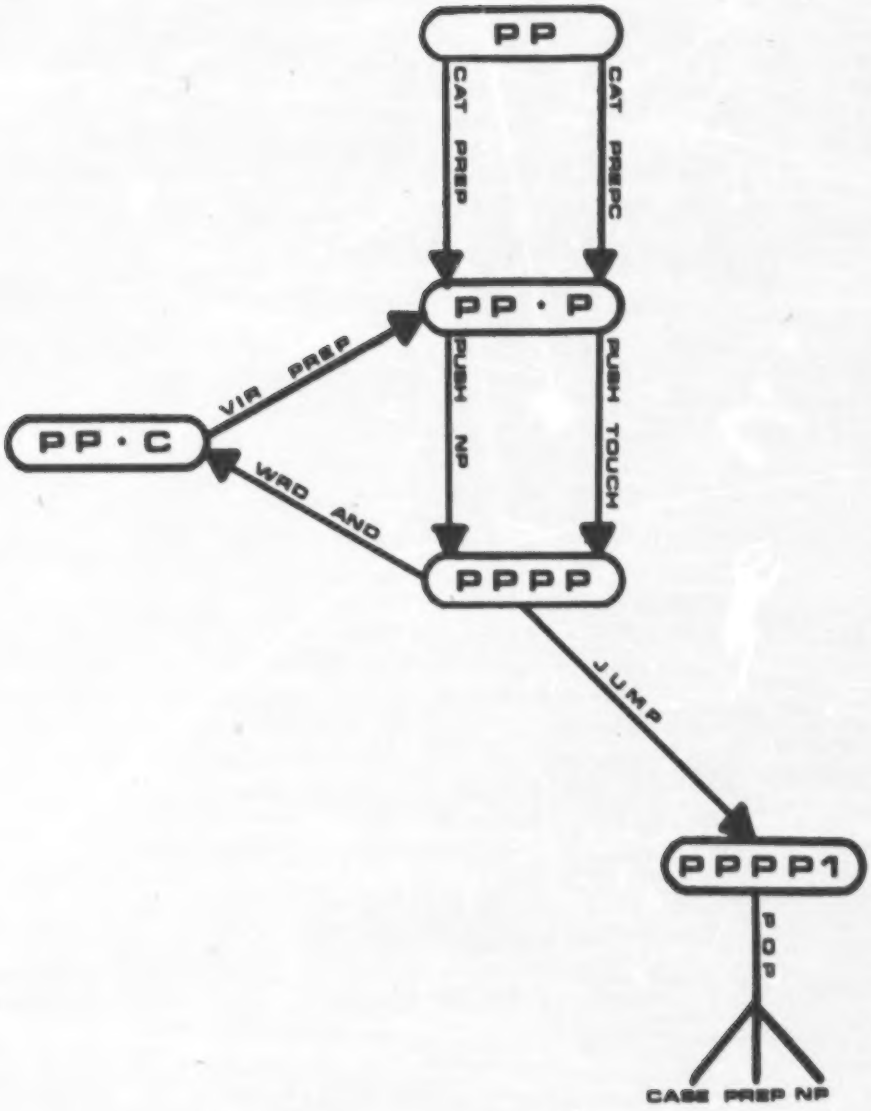
Appendix E: Sentence-Level Grammar



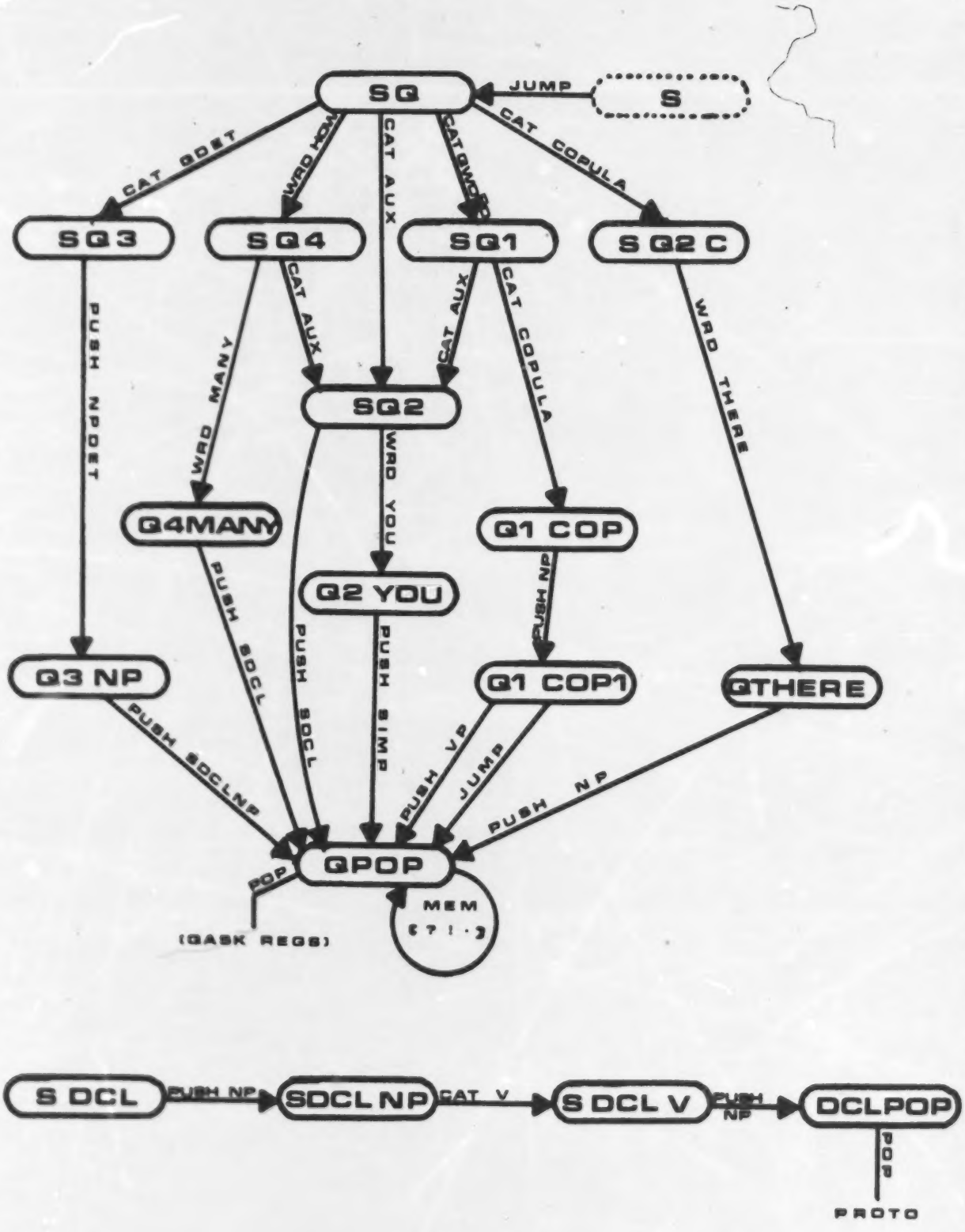
Appendix F: Touch Grammar

Appendix G: Noun Phrase Grammar





Appendix H: Prepositional Phrase Grammar



Appendix I: Question Grammar

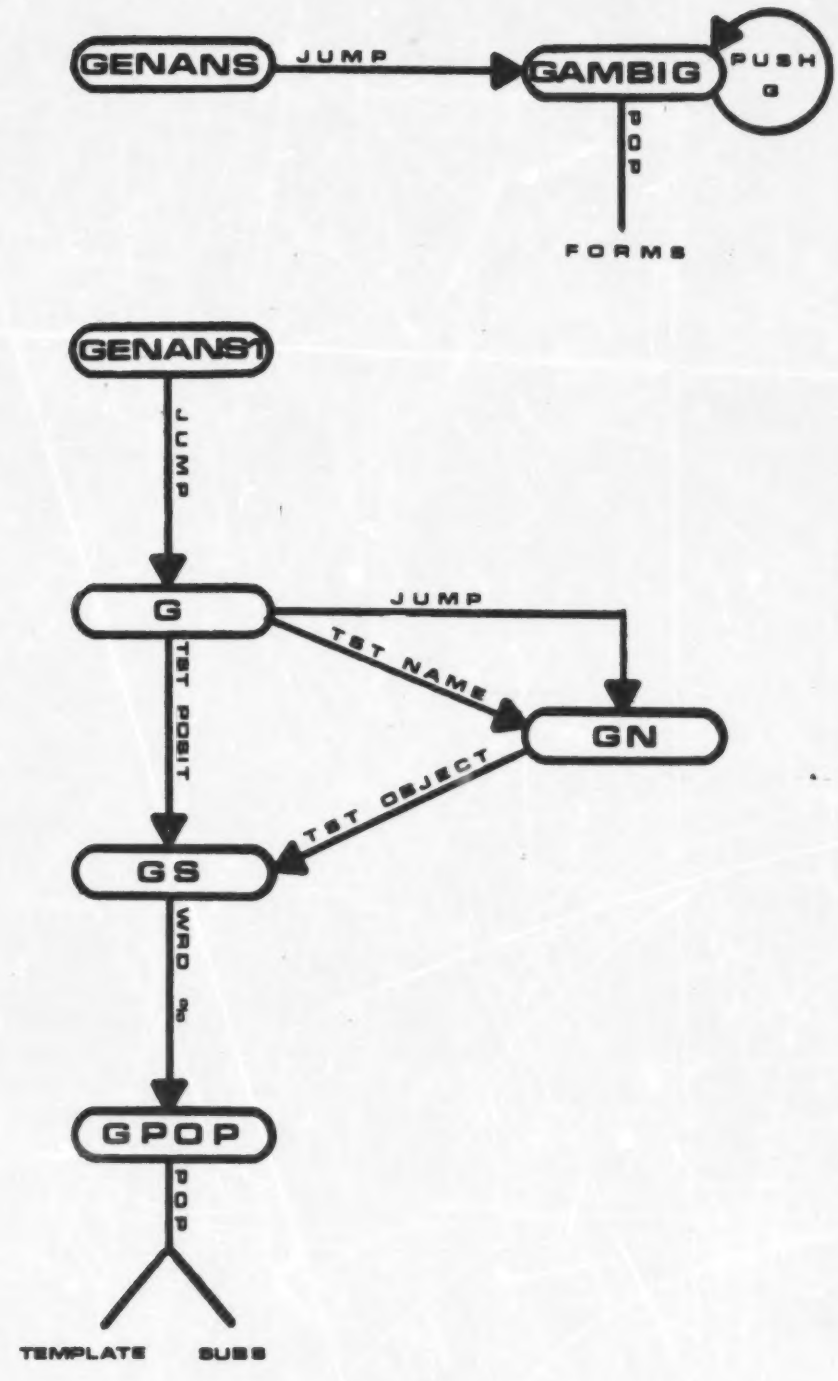
SENTENCES	TIME (ms.)
Draw a straight line from here <T> to there <T>	283
Please make from here <T> a line to there <T>	366
Please make me a line from (234, 412) to here <T>	366
Please erase the line here <T>	150
Draw a line between here <T> and here <T>	300
Put a point here <T>	116
Draw a point named FRED at <T>	217
Draw the point ETHEL here <T>	200
Now connect FRED and ETHEL	167
Draw a straight line from <T> through <T>	317
Join the point FRED with this point <T>	266
Stop	17
Please connect this <T> with this <T>	266
From here <T> draw a line to there <T>	333
From this point <T> draw a line to FRED	433
<T> <T> Draw a straight line from here to here	917
With this <T> connect this <T>	333
Draw a three inch line named L3 here <T>	217
Draw a horizontal line with a 3 inch length here <T>	350
Draw a vertical line 5 inches long called L5 to here <T>	317
Draw a circle with a 2 inch radius here <T>	300
Please draw a hundred mm line named DAVE at an angle of fifty degrees here <T>	401
Make me a circle called HEAD with a 2 inch diameter there <T>	266
Erase the line named ARM	167
Erase the line from here <T> to here <T>	316
Draw a circle of radius four units named EYE here <T>	250
Erase this circle <T>	133
Call this <T> X43	150
Name the three inch line L3	216
What can you draw?	67
Can you draw circles?	100
Can you draw a circle with a 3 inch circumference?	317
How many screen units is an inch?	250
Is there a point named P?	133
What did you draw at P?	183
Can you draw?	100

```
<message>          ::= (KB LA #Q (MG <prototype>))
<prototype>        ::=  ( (%TYPE    .   <type> )
                          (%ACTION  .   <act>  )
                          (%OBJECT  .   <obj>  )
                          (%NAME    .   <name> )
                          (%ORIENT  .   <ornt> )
                          (%POSIT   .   <posit>)
                          (%SIZE    .   <size> )  )

<type>             ::= IMP or Q

<act>              ::= DRAW or ERASE or NAME

<obj>              ::= ST-LINE or CIRCLE or POINT

<name>             ::= a LISP atom

<ornt>             ::= a number in degrees

<posit>            ::= (<pt>1)

<pt>               ::= (ENDPOINT <coords>) or
                       (LOCATOR  <coords>) or
                       (TOUCH    <coords>)

<coords>           ::= (<x-val>    .   <y-val>) or
                       (NODE       .   <node identifier>)

<x-val>            ::= x-coordinate in screen units

<y-val>            ::= y-coordiante in screen units

<node identifier>  ::= the identifier of a node in the semantic network

<size>             ::= (RADIUS          .  <num>) or
                       (DIAMETER        .  <num>) or
                       (CIRCUMFERENCE   .  <num>) or
                       <num>

<num>              ::= a length in screen units
```

Appendix K: Description of the Prototype

TEMPLATE NUMBER	TEMPLATE
T0	()
T1	(OK)
T2	(Thanks * for the session)
T3	(Natural Language Graphics system)
T4	(NLG - Pilot System - summer, 1976)
T5	(Please rephrase the sentence)
T6	(Object as specified cannot be found)
T7	(*)
T8	(yes)
T9	(no)
T10	(I don't know)
T11	(*)
T12	(* and *)
T13	(* , * , and *)
T20	(at *)
T21	(* has no name)
T22	(a * named *)
T23	(a *)

Appendix L: Language Generation Templates



Appendix M: Generative ATN Grammar

NAME	PART OF	FUNCTION
ACT-1	POINT	How to draw a POINT
ACT-2	ST-LINE	Structural check
ACT-3	--	---
ACT-4	--	---
ACT-5	ST-LINE	How to find midpoint
ACT-6	--	---
ACT-7	--	---
ACT-8	ST-LINE	How to draw a ST-LINE
ACT-9	--	---
ACT-10	--	---
ACT-11	--	---
ACT-12	--	---
ACT-13	POINT	Structural check
ACT-14	DEFPOINT	Structural check
ACT-15	--	---
ACT-16	DISTANCE	Structural check
ACT-17	ANGLE	Structural check
ACT-18	--	---
ACT-19	CIRCLE	Structural check
ACT-20	CIRCLE	How to draw a CIRCLE
ACT-21	CIRCLE	How to find a locator
ACT-22	CIRCLE	How to find a radius

Appendix N: List of Actions Used in Semantic Network

Appendix O: KBNET - Initial State of KB Network

```
((DEFINE
        SUBSETOF SUBSETOF-
        ELMNTOF ELMNTOF-
        NAME NAME-
        TYPE TYPE-
        ELMNTDESCR ELMNTDESCR-
        DATTR DATTR-
        ROLE ROLE-
        RESTR RESTR-
        ACTION ACTION-
        ATTR ATTR-
        INSTNCOF INSTNCOF-
        VAL VAL-
        CASE CASE-
        NUMBER NUMBER-
        OBJECT OBJECT-
        ERASED ERASED-
        STRUCTURAL STRUCTURAL-
        INSTANTIATES INSTANTIATES-
))
((BUILD TYPE SYSTEM NAME SCREENMODEL))
((BUILD NAME OBJECTS TYPE SET)=  XX)
((BUILD NAME LINES TYPE SET SUBSETOF *XX)= XY)
((BUILD NAME PRIMITIVES TYPE SET SUBSETOF *XX)= XZ)
((BUILD NAME DEFPOINTS TYPE SET SUBSETOF *XX))
((BUILD NAME ANGLES TYPE SET SUBSETOF *XX))
((BUILD NAME DISTANCES TYPE SET SUBSETOF *XX))
((BUILD NAME CURVES TYPE SET SUBSETOF *XY))
((BUILD NAME ST-LINES TYPE SET SUBSETOF *XY SUBSETOF *XZ))
((BUILD NAME POINTS TYPE SET SUBSETOF *XZ SUBSETOF (FIND NAME DEFPOINTS)
))
((BUILD NAME UNIVERSE TYPE SET
        SUBSETOF-  *XX
        SUBSETOF-  (BUILD NAME SITUATIONS
                          TYPE SET
                          SUBSETOF- (BUILD NAME CONNECTIONS TYPE SET)
                   )
))
((BUILD NAME DEFPOINT
        TYPE CONCEPT
        ELMNTDESCR-  (FIND NAME DEFPOINTS)
        STRUCTURAL ACT-14
        DATTR  (BUILD  ROLE XVALUE
                       TYPE DESCRIPTION
                       RESTR (FIND NAME DISTANCES)
               )
        DATTR  (BUILD  ROLE YVALUE
                       TYPE DESCRIPTION
                       RESTR (FIND NAME DISTANCES)
               )
))
((BUILD NAME DISTANCE
        TYPE CONCEPT
        ELMNTDESCR- (FIND NAME DISTANCES)
        STRUCTURAL ACT-16
))
```

```
((BUILD NAME ANGLE
        TYPE CONCEPT
        ELMNTDESCR- (FIND NAME ANGLES)
        STRUCTURAL ACT-17
))
((BUILD NAME POINT
        TYPE CONCEPT
        ELMNTDESCR- (FIND NAME POINTS)
        STRUCTURAL ACT-13
        DATTR (BUILD ROLE LOCATOR
                     TYPE DESCRIPTION
                     CASE %POSIT
                     RESTR (FIND NAME DEFPOINTS)
              )
        DATTR (BUILD ROLE TODRAW
                     TYPE DESCRIPTION
                     ACTION ACT-1
                     CASE %HOW
              )
))
((BUILD NAME ST-LINE
        TYPE CONCEPT
        ELMNTDESCR- (FIND NAME ST-LINES)
        STRUCTURAL ACT-2
        DATTR (BUILD ROLE ENDPOINT
                     TYPE DESCRIPTION
                     RESTR (FIND NAME DEFPOINTS)
              )
        DATTR (BUILD ROLE ENDPOINT
                     TYPE DESCRIPTION
                     RESTR (FIND NAME DEFPOINTS)
              )
        DATTR (BUILD ROLE LOCATOR
                     TYPE DESCRIPTION
                     CASE %POSIT
                     RESTR (FIND NAME DEFPOINTS)
                  )
        DATTR (BUILD ROLE MIDPOINT
                     TYPE DESCRIPTION
                     RESTR (FIND NAME DEFPOINTS)
                     ACTION ACT-5
              )
        DATTR (BUILD ROLE ORIENTATION
                     TYPE DESCRIPTION
                     CASE %ORIENT
                     RESTR (FIND NAME ANGLES)
              )
        DATTR (BUILD ROLE LENGTH
                     TYPE DESCRIPTION
                     CASE %SIZE
                     RESTR (FIND NAME DISTANCES)
              )
        DATTR (BUILD ROLE TODRAW
                     TYPE DESCRIPTION
                     CASE %HOW
                     ACTION ACT-8
              )
))
```

```
((BUILD NAME CIRCLES
        TYPE SET
        SUBSETOF (FIND NAME CURVES)
        ELMNTDESCR (BUILD NAME CIRCLE
                          TYPE CONCEPT
                          STRUCTURAL ACT-19
                          DATTR (BUILD ROLE LOCATOR
                                       TYPE DESCRIPTION
                                       ACTION ACT-21
                                       CASE %POSIT
                                       RESTR (FIND NAME DEFPOINTS)
                                 )
                          DATTR (BUILD ROLE RADIUS
                                       TYPE DESCRIPTION
                                       ACTION ACT-22
                                       CASE %SIZE
                                       RESTR (FIND NAME DISTANCES)
                                 )
                          DATTR (BUILD ROLE CIRCUMFERENCE
                                       TYPE DESCRIPTION
                                       RESTR (FIND NAME DISTANCES)
                                 )
                          DATTR (BUILD ROLE DIAMETER
                                       TYPE DESCRIPTION
                                       RESTR (FIND NAME DISTANCES)
                                 )
                          DATTR (BUILD ROLE TODRAW
                                       TYPE DESCRIPTION
                                       CASE %HOW
                                       ACTION ACT-20
                                 )
                    )
))

STOP  (*** TO FORCE WAY OUT OF MENTAL READ LOOP ***)
```

Note: The inverses of the links are indicated by the addition of a dash.
Note: The above description is in the form read by the MENTAL interpreter.

Appendix P: The GR Module Interface

The interface at GR is defined as follows:

```
<message> ::=   (GR KB #Q ( <prototype> ))

<prototype> ::= (   (%TYPE   . some type   )
                    (%ACTION . some action )
                    (%OBJECT . some object )
                    (%NAME   . some name   )
                    (%HOW    . <program>   )   )

<program> ::=    (<step>1)
<step>    ::=    (F4PNT x1 y1)
             or
                (F4LINE1 x1 y1 x2 y2)
             or
                (F4LINE2  x1 y1 angle length)
             or
                (F4LINE3 midx midy x2 y2)
             or
                (F4LINE4 midx midy angle length)
             or
                (F4CRCL cx cy radius)
             or
                (F4ERSE)
```

Appendix Q:

The Syntax of Calls to the QA Section of the KB Module

<message> ::= (KB LA #Q (QA <function>))

<function> ::= (INFO <keywords> <prototype form>)
or
(FINDOBJ <name> <type>)
or
(CONVERT a number <units>)
or
(FINDDRAW <object>)

<keywords> ::= (<keys>$_1$)

<keys> ::= %OBJECT or %NAME or %POSIT or NODE

<prototype form> ::= any valid complete or partial prototype

<name> ::= an object name extracted from an input sentence

<type> ::= ST-LINE or POINT or CIRCLE or NIL

<units> ::= UNIT or INCH or CM or MM or DEGREE or RADIAN

<object> ::= a primitive or suspected object